Nieuwsbrief van de Nederlandse Vereniging voor Theoretische Informatica

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

Inhoudsopgave

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

1 Van de Redactie

Beste NVTI leden,

Met genoegen bieden wij u bij deze de 9e NVTI Nieuwsbrief aan (het eerste lustrum is volgend jaar een feit). Hoewel de redactie de nodige personele wijzigingen heeft ondergaan, is de opzet zoals u die gewend bent. Naast een overzicht van het huidige bestuur, een aankondiging van de jaarlijkse NVTI Theoriedag met samenvattingen van de lezingen, bevat dit nummer nieuws van de onderzoeksscholen, en een aantal wetenschappelijke bijdragen. Bij deze onze hartelijke dank aan de auteurs van de diverse bijdragen! Deze Nieuwsbrief is, net als de Theoriedag, niet mogelijk zonder de steun van NWO-EW, Elsevier Publishing Company en de onderzoeksscholen SIKS, OzsL, en IPA. Namens de NVTI-leden onze hartelijke dank daarvoor. Verder is de de ondersteuning van het CWI van groot belang, met name bij het vervaardigen van de nieuwsbrief. Tenslotte willen wij de afgetreden redactieleden Mieke Brune, Jan-Willem Klop en Jan Rutten enorm bedanken voor hun inzet voor de NVTI, en de Nieuwsbrief in het bijzonder.

De redactie,

Susanne van Dam (Susanne.van.Dam@cwi.nl)
Joost-Pieter Katoen (katoen@cs.rwth-aachen.de)
Joost Kok (joost@liacs.nl)
Jaco van de Pol (Jaco.van.de.Pol@cwi.nl)
Femke van Raamsdonk (femke@cs.vu.nl)

2 Samenstelling Bestuur

Prof. dr. Jos Baeten (TU/e)
Dr. Hans Bodlaender (UU)
Prof. dr. Harry Buhrman (CWI en UvA)
Prof. dr. ir. Joost-Pieter Katoen (RWTH en UT)
Prof. dr. Joost Kok (RUL) voorzitter
Prof. dr. John-Jules Meyer (UU)
Dr. Jaco van de Pol (CWI en TU/e) secretaris
Dr. Femke van Raamsdonk (VU)
Prof. dr. Grzegorz Rozenberg (RUL)
Prof. dr. Gerard Renardel de Lavalette (RUG)
Dr. Leen Torenvliet (UvA)

3 Van de voorzitter

Geacht NVTI-lid,

Het is mij een genoegen om als nieuwe voorzitter van de NVTI een stukje voor de nieuwsbrief te schrijven.

Veel is nieuw bij de NVTI maar veel ook niet. Er is een nieuw “jong” bestuur dat met veel energie aan de slag is gegaan. Tegelijkertijd is voor de theoriedag het oude vertrouwde format gevolgd: vier prominente sprekers langs twee dimensies: twee uit het buitenland en twee uit het binnenland, twee sprekers over algoritmen en complexiteit en twee sprekers over semantiek en logica.

Op dit moment wordt de website van de vereniging (www.cwi.nl/orgs/NVTI/) vernieuwd. In Google staan we nu nog op plaats twee, maar we gaan de strijd met het National Veterans' Training Institute aan!

Ik hoop dat iedereen in de gelegenheid is om naar de theoriedag te komen. Sponsors (NWO, Elsevier, onderzoekscholen OzL, IPA, SIKS) maken de theoriedag weer mogelijk. En als u gaat, maak ook reclame in uw omgeving: wie weet zijn er wel een paar collega's die theorie een warm hart toedragen en graag meegaan!

Joost Kok, voorzitter NVTI

4 Theoriedag 2005

Theoriedag 2005 van de NVTI
(Nederlandse Vereniging voor Theoretische Informatica)

Vrijdag 4 maart 2005
Vergadercentrum Hoog Brabant, Utrecht

Het is ons een genoegen u uit te nodigen tot het bijwonen van de Theoriedag 2005 van de NVTI, de Nederlandse Vereniging voor Theoretische Informatica, die zich ten doel stelt de theoretische informatica te bevorderen en haar beoefening en toepassingen aan te moedigen. De Theoriedag 2005 zal gehouden worden op vrijdag 4 maart 2005, in Vergadercentrum Hoog Brabant te Utrecht, gelegen in winkelcenrum Hoog Catherijne, op enkele minuten loopafstand van CS Utrecht, en is een voortzetting van de reeks jaarlijkse bijeenkomsten van de NVTI die tien jaar geleden met de oprichtingsbijeenkomst begon.

Evenals vorige jaren hebben wij een aantal prominente sprekers bereid gevonden deze dag gestalte te geven met voordrachten over recente en belangrijke stromingen in de theoretische informatica. Naast een wetenschappelijke inhoud heeft de dag ook een informatief gedeelte, in de vorm van een algemene vergadering waarin de meest relevante informatie over de NVTI gegeven zal worden.

Programma (samenvattingen volgen beneden)

9.30-10.00:	Ontvangst met koffie
10.00-10.10:	Opening
10.10-11.00:	Lezing Prof. Dr. F. van Harmelen (VU) Titel: Anytime logical inference: Better half an answer in time than a perfect answer too late
11.00-11.30:	Koffie
11.30-12.20:	Lezing Prof. Dr. A.K. Lenstra (Bell Labs en TU/e) Titel: Information security, cryptology and factorizing
12.20-12.50:	Korte Presentaties
12.50-14.10:	Lunch (Zie beneden voor registratie)
14.10-15.00:	Lezing Prof. Dr. R. Cramer (CWI en Universiteit Leiden) Titel: Compressed pseudo-random secret sharing and its applications to practical secure multi-party computation
15.00-15.20:	Thee
15.20-16.10:	Lezing Dr. E. Meijer (Microsoft Research) Titel: Helping Programmers Program Better: Transferring Theory into Practice and Back
16.10-16.40:	Algemene ledenvergadering NVTI

Lunchdeelname

Het is mogelijk aan een georganiseerde lunch deel te nemen; hiervoor is aanmelding verplicht. Dit kan per email of telefonisch bij Susanne van Dam (susanne@cwi.nl, 020-592 4189), tot een week voor de bijeenkomst (25 februari). De kosten kunnen ter plaatse voldaan worden; deze bedragen ongeveer Euro 15. Wij wijzen erop dat in de onmiddellijke nabijheid van de vergaderzaal ook uitstekende lunchfaciliteiten gevonden kunnen worden, voor wie niet aan de georganiseerde lunch wenst deel te nemen.

Lidmaatschap NVTI

Alle leden van de voormalige WTI (Werkgemeenschap Theoretische Informatica) zijn automatisch lid van de NVTI geworden. Aan het lidmaatschap zijn geen kosten verbonden; u krijgt de aankondigingen van de NVTI per email of anderszins toegestuurd. Was u geen lid van de WTI en wilt u lid van de NVTI worden: u kunt zich aanmelden bij Susanne van Dam (susanne@cwi.nl), met vermelding van de relevante gegevens (naam, voorletters, affiliatie indien van toepassing, correspondentieadres, email, URL, telefoonnummer).

Steun

De activiteiten van de NVTI worden mede mogelijk gemaakt door de ondersteuning (financieel en anderszins) van de volgende instellingen: NWO/EW, CWI, Onderzoeksscholen OzsL, IPA en SIKS, Elsevier Science B.V.

Samenvattingen van de lezingen

Informatie beveiliging, cryptologie, en factorizeren
Spreker: Arjen K. Lenstra (Bell Labs en TUe)

Door het toenemend gebruik van electronische communicatie wordt informatie beveiliging een steeds populairder onderwerp. Dat is koren op de molen van wetenschappers die zich graag met mogelijk verwant onderzoek bezig houden. In deze voordracht belicht ik de mate van belang van een van die onderzoeksgebieden, cryptologie, en ga vervolgens nader in op de voortgang op het gebied van integer factorizatie methoden.

Compressed pseudo-random secret sharing and its applications to practical secure multi-party computation
Spreker: Ronald Cramer (CWI and Mathematical Institute, Leiden University)

In this talk we argue that pseudo-random secret sharing schemes can be used to limit round complexity of secure multi-party computation protocols. At the same time we ensure that communication complexity does not increase in the process, by employing new compression techniques for pseudo-random secret sharing. A main application of these techniques is a non-interactive threshold version of the Cramer-Shoup encryption scheme.
Joint work with Ivan Damgaard (Aarhus University) and Yuval Ishai (Technion, Haifa)

Anytime logical inference: Better half an answer in time than a perfect answer too late
Spreker: Frank van Harmelen (Vrije Universiteit, Amsterdam)

Classical logical inference characterises perfect modes of reasoning. Inference procedures give either perfect results or no results at all. The implementations of such inference procedures only give answers at the very end of their computation with no intermediate "currently best available answer".

For many reasons (both practical and theoretical), it would be attractive to have notions of logical inference which can provide approximate answers, which can compute such answers incrementally, and which allow an anytime trade-off between cost and quality.

In this presentation, I will first motivate the need for such approximate and anytime logics, I will describe a few proposals for such logics, and illustrate their use in various inference problems.

Helping Programmers Program Better: Transferring Theory into Practice and Back
Spreker: Erik Meijer (Microsoft)

As Donald Knuth remarks "Theory and practice are not mutually exclusive; they are intimately connected. They live together and support each other." This is (or should be) especially true in the context of programming languages. About four years ago, I left research and moved to work for Microsoft (product group, not research) with the hope to transfer programming language and type theory to practice and to find new inspiration for theory from real problems that occur in practice.

In this talk, I will report about my (first-hand) experiences so far, of how within Microsoft theory is applied in real shipping software ranging from the low-level execution engine/virtual machine (the CLR), to libraries/APIs (the .Net framework), up to the level of programming languages (C#, Comega, and Visual Basic). Examples will include security, concurrency, generics, higher-order functions, type-system unification of relations, XML and objects, and in particular the efficient implementation and formalization of co-inductive lazy imperative streams.

The talk will be guaranteed 100% free of marketing bullshit, but it will contain both code *and* greek symbols and horizontal lines.

5 Mededelingen van de onderzoekscholen

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. 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.

On the European front, IPA cooperates with the research schools BRICS (Denmark), TUCS (Finland), UKII (United Kingdom), IP (Italy), GEFI (Germany), and FI (France) in the European Educational Forum (EEF).

New scientific director After more than 8 years in office, Jos Baeten has stepped down as scientific director of IPA. Jos is one of the founding fathers of IPA, and under his supervision the Institute has grown to encompass 25 research groups (of eight Dutch universities and the CWI in Amsterdam), with over 250 people participating, including around a hundred Ph.D. students. IPA is very grateful for everything Jos Baeten has done to establish and develop the Institute.

As of May 1, Mark de Berg is the new scientific director of IPA. Mark is head of the Algorithms Groups of the Department of Mathematics and Computer Science of the TU/e. He received the prestigious VICI-grant of the NWO for his innovative work in computational geometry.

New research group joins IPA This June, the research group Biomodeling and Informatics (BMI) of the Department of Biomedical Engineering of the Technische Universiteit Eindhoven joined IPA. The group, founded by Peter Hilbers in 2001, focusses on the modeling of processes in living systems. Besides the development of methods and tools for biomedical modeling, it is concerned with building specific biomedical models and implementing them by means of algorithms and simulations. IPA welcomes the BMI group, which will enrich the Institute's field of research.

Activities in 2004

IPA has two multi-day events per year, the Lentedagen and the Herfstdagen, which focus on a particular subject. In the 2002 - 2006 period, each of the Herfstdagen will be dedicated to one of IPA's four so-called application areas: Networked Embedded Systems, Security, Intelligent

Algorithms, and Compositional Programming Methods. In 2004, the Herfstdagen were dedicated to Intelligent Algorithms, and the Lentedagen were on Hybrid Systems.

Lentedagen *April 14 - 16, De Kapellerput, Heeze*
More and more systems emerge in which discrete digital controllers control continuous physical processes. Such systems are called 'Hybrid Systems', because they combine discrete and continuous behaviour. Although there are methods for understanding both forms of behaviour separately (continuous behaviour in control science and system theory, and discrete behaviour in computer science), the proper functioning of a hybrid system critically depends on their interaction. Hence, the correct and efficient design of such systems requires expertise that is spread over different research and engineering communities.

Over the past few years, IPA-researchers working in the area of embedded systems have made contact with researchers from the communities studying behaviour of continuous systems, and started to create a common theoretical basis for the understanding of hybrid systems. The IPA Herfstdagen presented the research currently performed in and around IPA on formalisms, tools, and techniques for the modeling, analysis, validation, and verification of hybrid systems. The program was composed by Jozef Hooman (ESI/KUN), Rom Langerak (UT), Michel Reniers (TU/e), in cooperation with Arjan van der Schaft (UT, research school DISC). Abstracts, hand-outs, and papers, are available through the website at `www.win.tue.nl/ipa/archive/springdays2004/`.

Herfstdagen *November 22-26, Tulip Inn, Callantsoog*
Algorithms are vital building blocks for many software systems. The ever widening range of application for systems with algorithmic components in both industry and science brings different requirements to the fore than those traditionally studied in algorithmics research. For instance, algorithmic systems can be required to be 'always on', to be aware of their (unpredictable) surroundings, or to adapt their behaviour to that of their users over time. The Herfstdagen aimed to provide an overview of research in and around IPA on algorithms with these and other 'intelligent' properties. The program was composed by Emile Aarts (TU/e, Philips Research), Joost Kok (UL), and Jan van Leeuwen (UU).

The various sessions in the program addressed application domains that inspire new approaches, concepts and techniques because of the requirements algorithms have to meet. In Sensor Networking, for instance, individual sensor nodes need to become aware of where they are in relation to other nodes to build a communications network, and this network has to be able to adapt to changing conditions. In Ambient Intelligence, services are developed that try to optimize the experience of the user, rather than a system parameter, by adapting to the user's sensory capabilities and personal preferences. In the construction of Software Agents for negotation situations, algorithms are used to provide the negotiation strategies, drawing inspiration from game theory and economic theory. Other sessions were dedicated to data mining, machine learning, the semantic web, and human-algorithm interaction. Keynote speakers were: Roger Wattenhofer (ETHZ), Hannu Toivonen (Helsinki) and Berthold Vöcking (RWTH Aachen). Abstracts, hand-outs, and papers are available through the website at `www.win.tue.nl/ipa/archive/falldays2004/`.

In addition to organising the Lentedagen and Herfstdagen, IPA sponsored three further events: the 8th Dutch Proof Tools Day (July 9, University of Nijmegen), the first international conference on Quantitative Evaluation of Systems (QEST'04, September 27 - 30, University of Twente), and the third international symposium on Formal Methods for Components and Objects (FMCO 2004, November 2 - 5, Lorentz Center, Leiden University).

IPA Ph.D. Defenses in 2004

Georgi Jojgov (TU/e, 5 April), *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving.* Promotores: prof.dr. J.C.M. Baeten, prof.dr. H.P. Barendregt. Co-promotor: dr. J.H. Geuvers. IPA-Dissertation Series 2004-02.

Wil Michiels (TU/e, 8 April), *Performance Ratios for the Differencing Method.* Promotores: prof.dr. E.H.L. Aarts, prof.dr. J. van Leeuwen. Co-promotor: dr.ir. J.H.M. Korst. IPA-Dissertation Series 2004-01.

Sebastian Maneth (UL, 27 May), *Models of Tree Translation.* Promotor: prof.dr. G. Rozenberg. Co-promotor: dr. J. Engelfriet. IPA-Dissertation Series 2004-04.

Yuechen Qian (TU/e, 2 June), *Data Synchronization and Browsing for Home Environments.* Promotores: prof.dr. L.G.M. Feijs, prof.dr. J.C.M. Baeten. Co-promotor: dr.ir. M.P. Bodlaender. IPA-Dissertation Series 2004-05.

Pierluigi Frisco (UL, 3 June), *Theory of Molecular Computing – Splicing and Membrane systems.* Promotor: prof.dr. G. Rozenberg. Co-promotor: dr. H.J.H. Hoogeboom. IPA-Dissertation Series 2004-03.

Falk Bartels (VUA, 3 June), *On Generalised Coinduction and Probabilistic Specification Formats – Distributive Laws in Coalgebraic Modelling.* Promotores: prof.dr. J.J.M.M. Rutten, prof.dr. J.C.M. Baeten. IPA-Dissertation Series 2004-06.

Luís Cruz-Filipe (KUN, 15 June), *Constructive Real Analysis: a Type-Theoretical Formalization and Applications.* Promotor: prof.dr. H.P. Barendregt. Co-promotor: dr. J.H. Geuvers. IPA-Dissertation Series 2004-07.

Enrico Gerding (TU/e, 6 July), *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals Strategies, and Business Applications.* Promotores: prof.dr.ir. J.A. La Poutré, prof.dr. H.M. Amman. IPA-Dissertation Series 2004-08.

Andres Löh (UU, 2 September 2004), *Exploring Generic Haskell.* Promotores: prof.dr. J.Th. Jeuring, prof.dr. S.D. Swierstra. IPA-Dissertation Series 2004-11.

Milad Niqui (KUN, 27 September 2004), *Formalising Exact Arithmetic: Representations, Algorithms and Proofs.* Promotor: prof.dr. H.P. Barendregt. Co-promotor: dr. J.H. Geuvers. IPA-Dissertation Series 2004-10.

Reinder Bril (TU/e, 29 September), *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets.* Promotores: prof.dr. E.H.L. Aarts, prof.dr. G. Fohler. Co-promotor: dr.ir. W.F.J. Verhaegh. IPA-Dissertation Series 2004-13.

Ingrid Flinsenberg (TU/e, 30 September), *Route Planning Algorithms for Car Navigation.* Promotores: prof.dr. E.H.L. Aarts, prof.dr. J. van Leeuwen. Co-promotor: dr. J.H. Verriet. IPA-Dissertation Series 2004-12.

Esko Dijk (TU/e, 6 October), *Indoor Ultrasonic Position Estimation Using a Single Base Station.* Promotores: prof.dr.ir. C.H. van Berkel, prof.dr.ir. J.W.M. Bergmans. Co-promotor: dr. R.M. Aarts. IPA-Dissertation Series 2004-16.

Floortje Alkemade (TU/e, 7 October), *Evolutionary Agent-Based Economy.* Promotores: prof.dr.ir. J.A. La Poutré, prof.dr. H.M. Amman. IPA-Dissertation Series 2004-15.

Nicolae Goga (TU/e, 7 October), *Control and Selection Techniques for the Automated Testing of Reactive Systems.* Promotores: prof.dr.ir. L.M.G. Feijs, prof.dr. H. Brinksma. Copromotor: dr. S. Mauw. IPA-Dissertation Series 2004-09.

Martijn Schrage (UU, 15 October), *Proxima - A presentation-oriented editor for structured documents.* Promotores: prof.dr. S.D. Swierstra, prof.dr. J.Th. Jeuring, prof. L.G.L.T. Meertens. IPA-Dissertation Series 2004-18.

Jun Pang (VUA, 26 October), *Formal Verification of Distributed Systems.* Promotor: prof.dr. W.J. Fokkink. IPA-Dissertation Series 2004-14.

Simona Orzan (VUA, 25 November), *On Distributed Verification and Verified Distribution.* Promotor: prof.dr. W. Fokkink. Copromotor: dr. J.C. van de Pol. IPA-Dissertation Series 2004-17.

Evgeny Eskenazi and Alexander Fyukov (TU/e, 6 December), *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures.* Promotores: prof.dr.ir. J.F. Groote, prof.dr.dipl.ing. D.K. Hammer. IPA-Dissertation Series 2004-19.

Barend van den Nieuwelaar (TU/e, 7 December), *Supervisory Machine Control by Predictive-Reactive Scheduling.* Promotores: prof.dr.ir. J.E. Rooda, prof.dr. J.C.M. Baeten.Copromotor: dr.ir. J.M. van de Mortel-Fronczak. IPA-Dissertation Series 2004-21.

Pieter Cuijpers (TU/e, 9 December), *Hybrid Process Algebra.* Promotores: prof.dr.ir. J.F. Groote, prof.dr.ir. P.P.J. van den Bosch. Co-promotor: dr.ir. M.A. Reniers. IPA-Dissertation Series 2004-20.

100th dissertation in series Nicolae Goga's thesis *Control and Selection Techniques for the Automated Testing of Reactive Systems* is the 100th title in the IPA dissertation series which openend with the thesis *The State Operator in Process Algebra* by Javier Blanco in 1996.

Activities in 2005

The themes for the major IPA-events of 2005 are known: the Lentedagen will be on Software Architecture and the Herfstdagen will be dedicated to Security. In addition, IPA is planning to stage two of its three Basic Courses this year. The Basic Courses are intended to provide Ph.D. students with an overview of IPA's major research fields: Algorithms and Complexity, Formal Methods, and Software Technology.

The first Basic Course and the Lentedagen have a definite schedule and location. More information on all upcoming activities will become available through the IPA web site.

Basic Course on Algorithms and Complexity *January 31 - February 4, TU/e.*
This Basic Course, which is hosted by IPA at the Technische Universiteit Eindhoven, focusses on five subjects areas in algorithmics, where succesfull research is being conducted by groups in IPA. From each of these areas, operations research, graph and network algorithms, natural computation, and alternative computational models, a topic is taken to which an entire course day is dedicated. The final course day will addres an interesting application area for algorithmic techniques, in this case bio informatics. See: `www.win.tue.nl/ipa/activities/algbasiccourse2005/`.

Lentedagen on Software Architecture *March 30 - April 1, De Korenbeurs, Made.*
This year's Lentedagen will be dedicated to software architecture, in particular for distributed applications. After having looked at component-based architectures (Herfstdagen 1999), object-oriented architectures (Lentedagen on UML, 2000), and peer-to-peer systems (Lentedagen on Middelware, 2002), this year's Lentedagen will address recent developments in this area such as service-oriented architectures. See: `www.win.tue.nl/ipa/activities/springdays2005/`.

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/`

The School for Information and Knowledge Systems (SIKS) in 2004

by Richard Starmans

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 particularly in the area of information and knowledge systems (IKS). SIKS is an interuniversity research school that comprises 12 research groups from 10 universities and CWI. When SIKS received its accreditation by KNAW in 1998, only 35 Ph.D. students and about 70 research fellows were involved in the school. Currently, over 300 researchers are active, including 140 Ph.D.-students. The Vrije Universiteit in Amsterdam is SIKS' administrative university. The office of SIKS is located at Utrecht University In June 2003 SIKS was reaccreditated by KNAW for another period of 6 years.

Activities

We here list the main activities (co-)organized or (co-)financed by SIKS in 2004. We distinguish basic courses, advanced courses and other acitivities (including masterclasses, workshops, oneday seminars, conferences and research colloquia)

Basic courses

Research methods and methodology for IKS, February 09-11, 2004 Het Bosgoed, Lunteren
Scientific directors: dr. H. Weigand (UvT), prof. dr. R. Wieringa (UT),
prof. dr. J-J.Ch. Meyer (UU), dr. R. Starmans (UU)

Interactive Systems , May 14 - 16, 2004, Huize Bergen, Vught
Scientific director: Prof. dr. G. van der Veer (VU)

Architectures for IKS, May 16 - 18, 2004, Huize Bergen, Vught
Scientific director: Prof. dr. E. Proper (RUN)

Information and Organisation, December 06-08, 2004, Huize Bergen, Vught
Scientific director: dr. H. Weigand (UvT)

Information Retrieval, December 08-10, 2004, Huize Bergen, Vught
Scientific director: prof. dr. T. van der Weide (RUN)

Advanced courses

Spring course on Datamining, April 13-17, 2004, Maastricht
Course directors: Dr. E. Smirnov (UM), Dr. J. Donkers(UM), Prof. dr. E.O. Postma (UM)

"Mobile Commerce" (m-Commerce) April 21-22, 2004, Amsterdam
Course directors: prof. dr. H. Akkermans (VU), Dr. N. Sadeh (VU)

"The semantic web", November 22 - 23, 2004, Conference center Woudschoten, Zeist.
Course directors: Prof. dr. F van Harmelen (VU), Prof. dr. G. Schreiber (VU)

Other activities

- SIKS/ICS Symposium on Agent Organizations, January 13, 2004, Utrecht
- Paradoxaal, een symposium over paradoxen, March, 03, 2004, Utrecht
- Theoriedag 2004 van de NVTI, March 05, 2004, Utrecht
- SIKS-day 2004, March 12 2004, Castle of Zeist, Zeist
- Workshop: Intelligent Risk Analysis & Management, May, 19, 2004, Rotterdam
- Workshop on XML Databases and Information Retrieval, June 21, 2004 Enschede
- Symposium "Agents Everywhere", July 02, 2004, Enschede
- Social Intelligence Design 2004, July 05-07, Enschede
- AH2004; conference on Adaptive Hypermedia 2004, August 23-26, 2004, Eindhoven
- ICEC 2004; conference on Entertainment Computing, September 01-03, 2004, Eindhoven
- SIKS/ICS Master Class on Agent Societies, October 13, 2004, Utrecht
- Symposium "AI in the Wild", October 20 2004, Groningen
- BNAIC 2004, October 21-22, 2004, Groningen
- SIKS/ICS Masterclass "Logic and Agents:It is all in the game", November 24, 2004 Utrecht
- SIKS Masterclass by Kalle Lyytinnen;"*Innovation in Software Development and Architecture*", November 29, 2004 Enschede
- SIKS-IKAT Research colloquium: organized 6 times in Maastricht
- CABS-SIKS Research colloquium, organized 7 times in Delft en Utrecht
- IKS-SIKS Information science seminar, organized 9 times in Utrecht

Doctoral dissertations in 2004

In 2004 20 researchers successfully defended their Ph.D.-thesis and published their work in the SIKS-dissertation Series.

Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic
Promotor: prof.dr. J.-J. Ch. Meyer (UU)
Co-promotor: dr. F. Dignum (UU), dr. H.Weigand (UvT)
Promotie: 12 January 2004

Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques
Promotor: prof.dr.ir. J.L.G. Dietz (TUD)
Promotie: 26 January 2004

Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business
Promotor:Prof.dr.ir. M.P. Papazoglou (UvT)
Co-promotor: dr.rer.nat. M.A. Jeusfeld (UvT)
Promotie: 20 February 2004

Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving
Promotor: prof. dr. F.A.H. van Harmelen (VU)
Co-promotor: dr. A.C.M. ten Teije (VU)
Promotie: 23 March 2004

Viara Popova (EUR)
Knowledge discovery and monotonicity
Promotor: prof.dr. A. de Bruin (EUR)
Co-promotor: dr. J.C. Bioch (EUR)
Promotie: 01 April 2004

Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures
Promotores: prof.dr. B.J. Wielinga (UvA), prof.dr.A.Th.Schreiber (VU)
Promotie: 06 April 2004

Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes
Promotores: Prof. dr. M.J.H. Meijer (UM), Prof. dr. H.J. van den Herik (UM)
Promotie: 13 May 2004

Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning
Promotor: Prof. dr. R.P. van de Riet (VU)
Co-promotor: dr. mr. H. Prakken (UU)
Promotie: 22 June 2004

The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents
Promotor: Prof. dr. ir. A. Nijholt (UT)
Co-promotor: dr. D.K.J. Heylen (UT)
Promotie: 1 July 2004

Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play
Promotores: Prof. dr. ir. A. Nijholt (UT), Prof.dr. W. van der Hoek (University of Liverpool)
Co-promotor: dr. J. Zwiers (UT)
Promotie: 1 July 2004

Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium
Promotores: Prof. dr. J.-J. Ch. Meyer (UU), Prof.dr. W. van der Hoek (University of Liverpool)
Co-promotor: dr. C. Witteveen (TUD)
Promotie: 6 September 2004

Michel Klein (VU)
Change Management for Distributed Ontologies
Promotores: prof dr. A.Th. Schreiber (VU), prof dr. J.M. Akkermans (VU)
Promotie: 14 September 2004

Joop Verbeek (UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politile gegevensuitwisseling en digitale expertise
Promotores: Prof. dr. H.J. van den Herik (UM), Prof.mr. Th.A. de Roos (UL)
Promotie: 14 Oktober 2004

Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects
Promotores: Prof. dr. R. de Hoog (UVA) Prof. dr. B.J Wielinga (UVA)
Promotie: 20 October 2004

Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning
Promotor: Prof. dr. A.E. Eiben (VU)
Co-promotor: dr. E. Marchiori (VU)
Promotie: 26 October 2004

Arno Knobbe (UU)
Multi-Relational Data Mining
Promotor: Prof. dr. A.P.J.M. Siebes (UU)
Promotie: 22 November 2004

Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval
Promotor: Prof. dr. F.M.G. de Jong (UT)
Co-promotor: Dr. A.P. de Vries (CWI)
Promotie: 25 November 2004

Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models
Promotor: Prof. Dr. B.J. Wielinga (UvA)
Co-promotor: Dr. B. Bredeweg (UvA)
Promotie: 29 November 2004

Mark Winands (UM)
Informed Search in Complex Games
Promotor: prof.dr. H.J. van den Herik (UM)
Co-promotor: dr. J.W.H.M. Uiterwijk (UM)
Promotie: 1 December 2004

Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams
Promotores: Prof dr W. Baets (Nyenrode), Prof. dr. G. van der Veer (VU)
Co-promotor: dr. Th. Homan (Nyenrode)
Promotie: 10 December 2004

The Theory of Tetris

Hendrik Jan Hoogeboom and Walter A. Kosters

Leiden Institute of Advanced Computer Science
Universiteit Leiden, The Netherlands
{hoogeboo,kosters}@liacs.nl

1 Introduction

Any algorithm requires a theoretical analysis. Such an analysis may address issues like complexity (e.g., NP-completeness [9]), decidability and practical properties concerning special cases. In this paper we would like to discuss the TETRIS game in this light. We will first describe the game and some of its variants, show NP-completeness of a certain decision problem naturally attached to the game and then prove (un)decidability in some other cases. We conclude with some practical topics that arise from the NP-completeness proof.

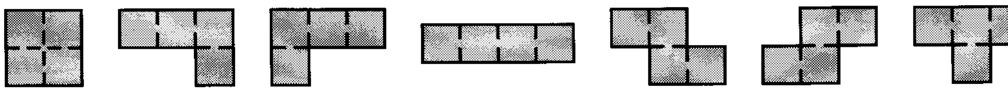
This paper is based on a series of articles that begins with the original NP-completeness proof of Demaine, Hohenberger and Liben-Nowell from MIT [7], that was well-noticed in the popular press. The proof was simplified in [3], leading to a joint journal paper [2]. In [13] and [14] the other issues mentioned above were dealt with. For full proofs we refer to these papers.

TETRIS is a one-person game where random pieces consisting of four blocks fall down, one at a time, in an originally empty rectangular game board. The player is allowed to rotate and horizontally move the falling piece. If an entire row of the board is filled with blocks, it is removed ("cleared"). The main purpose of the game is to keep on playing as long as possible.

The decision problem under consideration is essentially the following. Given a partially filled game board (referred to as a TETRIS configuration), and an ordered finite known series of pieces, is it possible to play in such a way that the whole board is cleared? The NP-completeness proof is by reduction. It is shown that instances of the so-called 3-PARTITION problem can be translated into rather involved TETRIS configurations, where solutions correspond with boards that can be cleared. The configurations used suggest the question of constructibility: which configurations can be reached during game play? The rather surprising answer appears to be that almost any configuration can be reached, given suitable pieces. Another issue has to do with decidability: if the user interaction is somewhat bounded, is it then decidable whether certain natural sets of piece sequences contain "clearing" sequences? All these topics will be addressed in the sequel.

2 Rules

The game of TETRIS is played on a rectangular board consisting of square (initially empty) cells. The board is of fixed width $w \geq 4$ and, for our purposes, of unbounded height. Seven game pieces can be used, each covering four board cells — from now on usually called blocks; they are depicted below. These pieces are known as (from left to right) O or *square*, J or *leftgun*, L or *rightgun*, I or *dash*, Z or *leftsnake*, S or *rightsnake*, and T or *tee*:



The "computer" generates a (usually random) sequence of pieces that drop down from the top of the board until they rest on top of (parts of) previously dropped pieces or on the bottom of the game board. The user/player can determine the position and orientation of the pieces by rotating and moving them horizontally while they fall. Whenever all the cells of a row (also called line) of the game board are occupied, the line is cleared; all blocks above it are lowered by one row (and no more). This row clearing can happen for several lines simultaneously.

In TETRIS the purpose usually is to clear as many rows as possible given the generated sequence of pieces, while avoiding to run out of space vertically. As the game of TETRIS itself is finite state (and hence decidable) when played on a board of given width and height, here we assume the board is of unbounded height.

TETRIS has some peculiar implementation details. Let us mention a few examples.

1. In the NP-completeness proof below it is essential that TETRIS pieces that touch the bottom of the game board or blocks from other pieces can still slide horizontally before they are "fixed". In some implementations this is however not possible, and pieces are then fixed immediately after touchdown. The NP-completeness proof might still hold for this other version, but a new construction is necessary, since the current one relies on filling overhangs with horizontally sliding squares.

2. In [7, 2] some attention is given to *rotation models*. It is indeed a problem, or rather a convention, to describe which "holes" allow TETRIS pieces to pass through, perhaps involving meticulous intermediate rotations. And when are pieces still allowed to rotate? In the sequel we do not refer to this issue anymore, since the constructions involved do not give rise to problems of this kind.

3. Some people are surprised by seeing floating blocks in TETRIS configurations. As will be shown later, (nearly) every configuration is constructible, i.e., can occur during regular game play. This includes situations where blocks do not rest on other blocks, but just remain floating — on air, so to speak. This is a consequence of the strictly applied rule that as one or more lines are cleared, they are removed from the game board; blocks above these disappearing lines precisely fall down as many lines as were cleared. This issue will only be of importance in the section on the construction of configurations.

3 NP-completeness

As mentioned in the introduction, we shall analyze the complexity of some decision problems related to TETRIS. We shall also loosely describe the proof of the main NP-completeness result. Precise definitions, theorems, proofs and related results can be found in [7, 3, 2].

In this section we consider the following decision problem, called TETRIS CLEARING:

Instance. A TETRIS game board partially filled with blocks, and an ordered sequence of TETRIS pieces.

Question. Is it possible to play in such a way that the game board is left empty in the end?

It is not difficult to see that this problem is in the class NP. We now prove NP-hardness. As mentioned before, the proof is by reduction. We use the 3-PARTITION problem:

Instance. An sequence $a_1, a_2, \ldots, a_{3s}$ of positive integers and a positive integer T, so that $T/4 < a_i < T/2$ for all i with $1 \leq i \leq 3s$ and so that $\sum_{i=1}^{3s} a_i = sT$.

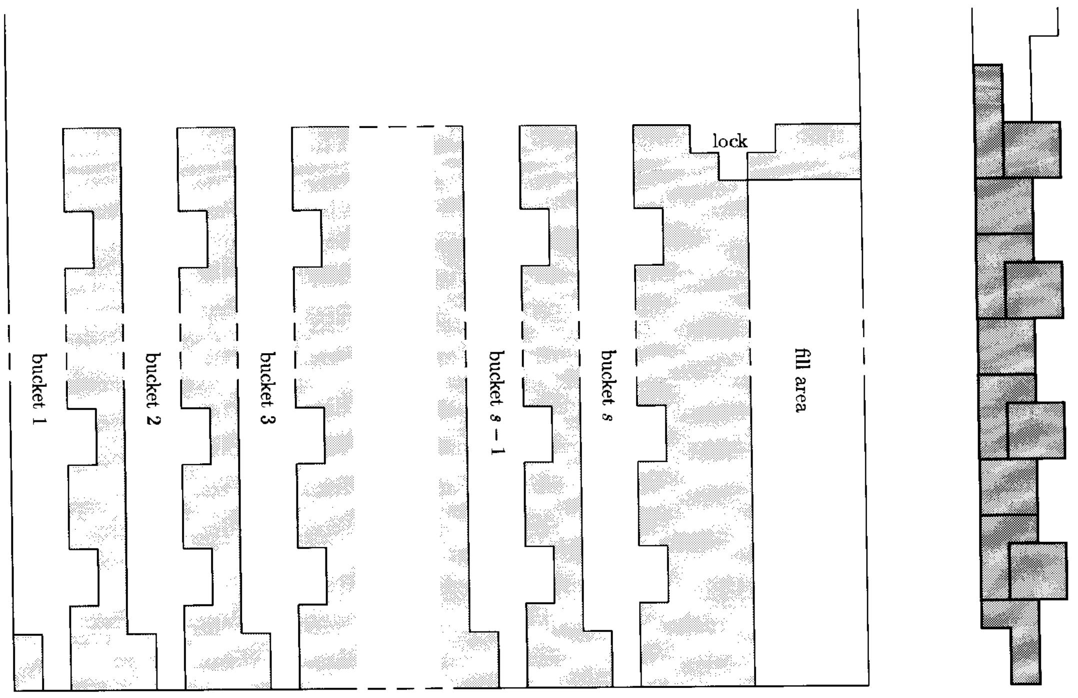
Question. Can $\{1, 2, \ldots, 3s\}$ be partitioned into s disjoint subsets $A_1, A_2, \ldots, A_s$ so that for all j with $1 \leq j \leq s$, we have $\sum_{i \in A_j} = T$?

In [9] it is shown that this problem is NP-complete *in the strong sense*, which means that it is NP-hard even if the inputs a_i and T are provided in unary.

Now the main result is:

Theorem. TETRIS CLEARING is NP-complete.

We give a brief sketch of the proof. We start from an instance of the 3-PARTITION problem. We then construct the following initial TETRIS game board (see picture below, left). The height of



the top row is $5T + 18$. We call the empty columns ***buckets*** and the big rectangular space on the right *fill area*; the T-shaped area above right is called the *lock*. Every bucket represents a subset as in 3-PARTITION. There are s buckets just like there are s subsets in 3-PARTITION.

The sequence of pieces for our game consists of a series of pieces for each a_i, followed by a number of pieces after all the a_i's. Each integer a_i is "coded" by one L, a_i times the triple O–J–O, and one pair O–I (see right part of the figure above for $a_i = 3$). The final pieces are: s successive L's to fill the buckets, one T to open the lock and exactly enough (i.e., $5T + 16$) successive I's to cover the fill area.

It is not much of a problem to show that the given sequence of pieces can clear the initial game board, in the case of a "yes" instance of 3-PARTITION. It is harder to prove that if the sequence cannot clear the board, the original instance could not fulfill the properties of a "no" instance. We just mention a few interesting details.

We suppose that we are looking at a sequence that can clear the original TETRIS game board. The volume of the pieces is precisely what is needed to fill the empty cells in this board. This implies — among other things — that no pieces are allowed to stick out above the original highest row. The fill area and the lock ensure that there will be no line clearings before all the buckets to the left are filled. Then comes the unique T piece that opens the lock in the upper right, after which a series of I's does the clearing. The main body of this part of the proof is devoted to showing that the series of pieces that "code" a number are precisely in this order required to fill the bucket, and cannot be spread over several buckets. Also notice the use of sliding O's to fill the buckets. □

4 Decidability

In this section we discuss (un)decidability issues related to the game of TETRIS, as reported in [13] (where details of the proofs can be found).

We consider different *models of user intervention*. On the one hand we have the normal TETRIS rules, as described above, where the user has many possibilities to influence the result. At the other extreme we have the model where the user is not allowed any intervention: once the "computer" fixes the piece, its orientation and its horizontal position, the piece drops down in the specified orientation, and in the specified position.

As for a given game board the number of initial possibilities of each piece — its orientation and the columns occupied — is bounded, the sequence of pieces dropped can be described by a string over a finite alphabet. This suggests the following decision problem, TETRIS *with Intervention Model M*:

Instance. A regular language L describing sequences of TETRIS pieces (with their initial orientation and horizontal position) for a given width game board.

Question. Is there a sequence in L that leaves the game board empty after dropping all the pieces into an initially empty game board, according to the "model" M? I.e., does the user have a way to clear the entire sequence, while adhering to the rules in M?

Note that if the user is not allowed any intervention (we call this the *Null Intervention Model*, and refer to the corresponding decision problem as TETRIS *without intervention*), there are no choices to be made. For more complicated models, we are looking for "optimal" user actions that lead to total clearings.

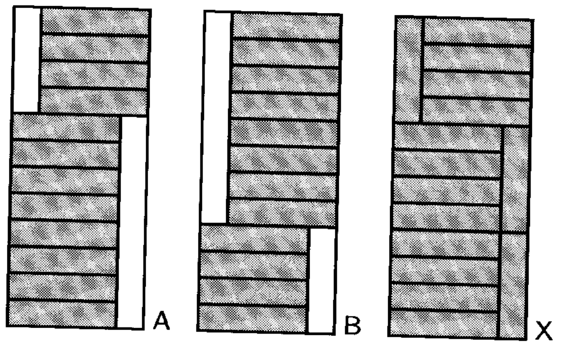
4.1 An undecidability result

We now have the following undecidability result:

Theorem. TETRIS without intervention, for sequences consisting only of I's on a board of width 10, is undecidable.

The proof is based on a reduction from the Post Correspondence Problem [15]. The basic idea is the following.

Given an instance of the PCP — two sequences $(u_1, \ldots, u_n)$ and $(v_1, \ldots, v_n)$ of strings over a two-letter alphabet $\{a, b\}$ — we construct an instance L of TETRIS without intervention, on a board of width 10. The left and right halves of the board (each a board of 5 cells wide) will act as stacks holding proposed solutions to the PCP, i.e., words of the form $u_{i_1} u_{i_2} \cdots u_{i_k} = v_{i_1} v_{i_2} \cdots v_{i_k}$ for some $k \geq 1$, and $1 \leq i_1, i_2, \ldots, i_k \leq n$. To build the contents of the stacks we need three basic piles of I's, that we call A, B and X. The first two of these represent the two symbols a and b of the alphabet of the PCP; the last one is used for padding the two copies of the solutions.



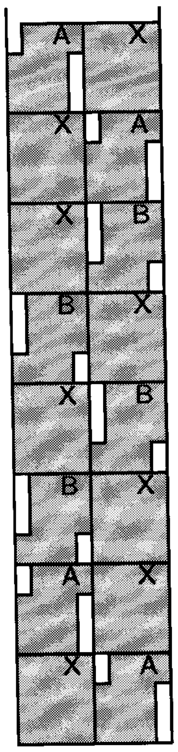
Note that the A and B piles can be removed (popped from the stack) following the rules of TETRIS, by dropping three vertical I's in the proper columns, provided the piles are next to an X pile on the other stack. The piles are designed in such a way that the vertical I's used to remove the blocks do not fall through to the next pile. Pieces dropped in the first column are blocked by the bottom rows of the pile, pieces falling in the fifth column are blocked by the topmost row of the pile below (or by the "floor").

Now first, the language L (or the corresponding finite automaton) prepares nondeterministically a sequence of piles, pushing onto the two stacks the same (nonempty) sequence of A's and B's, but randomly interleaved with X's. This part is independent of the particular instance of the PCP.

Then, in a second phase, L tries to clear the board, guessing a solution of the PCP, by repeatedly picking an index i $(1 \leq i \leq n)$ and trying to pop the left stack according to the string u_i and the right stack according to the string v_i.

The rest of the proof (see [13]) shows that the original PCP has a solution if and only if the language L has a way to leave the empty game board.

As an example, a configuration left after the first phase of our construction is depicted to the right: in the left stack we can read (top-to-bottom) a, b, ba while we encounter ab, b, a in the right stack. This corresponds with a solvable PCP. □



4.2 Some decidability results

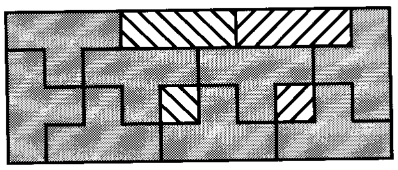
Quite amazingly, the undecidability result uses only a single type of piece. Let us now look at other ones. A simple argument shows that a nonempty sequence of either S or Z pieces cannot clear the board (cf. [5]), so the problem restricted to those pieces becomes trivially decidable. For the pieces T, L and J we can conceive a configuration that can be used to construct stacks, and similar arguments as for I hold (albeit on a board of width 16).

Finally, for O only very regular patterns are possible that leave an empty board. This is the basis for the following result:

Theorem. TETRIS without intervention, for sequences consisting only of O's on a board of width 10, is decidable. □

We reconsider the decision result above, now allowing user translation and rotation of the pieces that are specified by the sequences in the given regular language L. The intervention is just as in the standard TETRIS game. We refer to the corresponding decision problem as TETRIS *with normal intervention.*

The general question is related to the many tiling problems for polyominoes (see, e.g., [11, 8]), as a tiling of a rectangle by TETRIS tetrominoes implies a possible clearing of the board using the TETRIS pieces in some suitable order. However, apart from the fact that the TETRIS problem also deals with the *order* in which the pieces are offered, classical tiling is more restricted: it does not allow intermediate clearing of rows. As an example, ten T's can clear the TETRIS game board (of width 10, as below) whereas there is no tiling of the 10 by 4 rectangle using T's [16].



The sequences of the rectangular TETRIS pieces O and I that can be used to leave an empty game board have a simple characterization. Our result is valid for standard width 10, but can be stated slightly more generally. We need the following Lemma:

Lemma. A sequence of I's and O's can be dropped into an initially empty game board of width $4k + 2$ ($k \geq 1$) leaving the empty board if and only if the number of pieces is a multiple of $2k + 1$, and the number of I's is even. □

We have an immediate corollary:

Theorem. TETRIS with normal intervention, for sequences consisting only of I's and O's on a board of width 10, is decidable. □

There are connections with the strategies developed for winning two piece TETRIS games presented in [4].

Let us conclude this section with a slightly unexpected result. Restricted to a single piece (which can be other than the seven tetrominoes in standard TETRIS) TETRIS with normal intervention is decidable, even though we do not (need to) explicitly know the decision algorithm in each particular case:

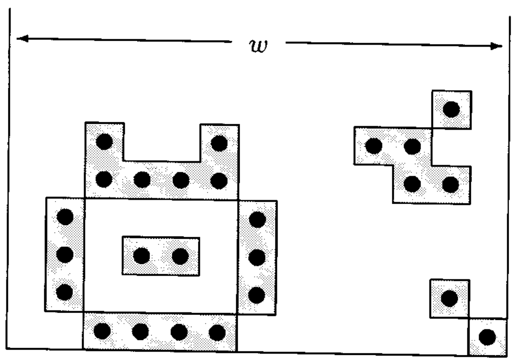
Theorem. TETRIS with normal intervention, for sequences consisting of copies a single fixed piece, on a board of fixed width, is decidable.

The proof relies on the fact that only the number of pieces matters, and that these numbers form a so-called semi-linear set [10]. □

5 Constructibility

The NP-completeness proof requires a rather intricate TETRIS configuration to start with. It seems a natural question to ask whether or not this configuration can occur during normal game play, and more general: what are the configurations that can occur? The answer is somewhat surprising: (almost) any configuration can show up!

A TETRIS *configuration* is a game board, where some of the cells are already occupied. A configuration is called *constructible* if it is possible to reach it, from an initially empty board, with a suitable series of pieces using appropriate translations and rotations. In this section we shall prove that essentially every reasonable configuration is constructible. The one non-trivial exception deals with boards of even width, where some simple parity condition should be fulfilled. The example configuration to the right, on a board of width $w = 13$, is constructible.



Our construction requires 276 TETRIS pieces, clearing $(4 \cdot 276 - 25)/13 = 83$ intermediate rows.

Let us first remark that if the width w of the board is a multiple of 4, at any time the number of blocks in the current configuration is a multiple of 4: each new piece adds 4 blocks, whereas a line clearing removes w blocks. So clearly, the number of blocks in any constructible configuration should then be a multiple of 4. Similarly, if $w + 2$ is a multiple of 4, the number of blocks should be a multiple of 2. These two simple restrictions appear to be the only ones:

Theorem. A configuration of p blocks is constructible using suitable TETRIS pieces starting from the empty board of width w if and only if

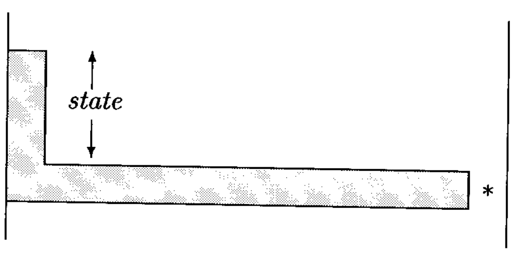
1. no row is completely full, and
2. no row below the highest one containing blocks is completely empty, and
3. if w is a multiple of 4, then so is p; if $w + 2$ is a multiple of 4, then p is even.

The next section is dedicated to the proof of the theorem, giving an explicit construction (see [12] for an implementation in the form of a Java-applet). In the sequel we shall assume that all three conditions mentioned in the theorem are met.

5.1 The construction

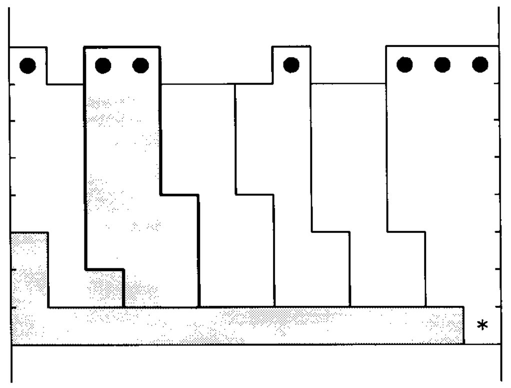
The configuration on the board is constructed row-by-row in a modular fashion. In [14] all details are given. For each row the construction consists of two phases.

First we build a *platform* that serves as a scaffolding for the construction work (it prevents TETRIS pieces from falling down to lower rows). In general the platform looks as follows, see the picture to the right. The $*$ denotes the bottom right empty square of the platform; once it is filled, its whole row will be cleared. The platform construction requires 3 or 7 intermediate rows that are cleared.

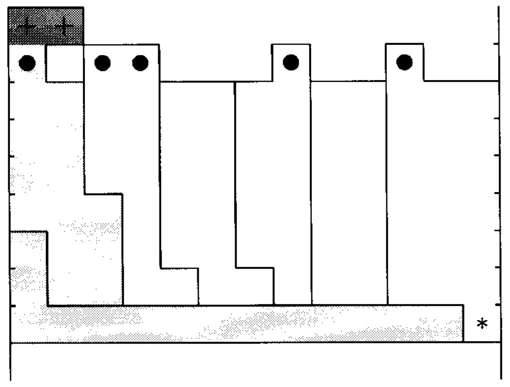


The number of squares sticking out vertically above the platform at the left end may vary between 0 and 3, and is referred to as the *state of the platform*. We need such a state as the total number of cells presently occupied or cleared in the past must be a multiple of 4.

In the second phase, the row construction phase, we build the blocks ● of the next row of the configuration on top of the platform, using six additional rows. Basically we construct consecutive "rectangles" two columns wide and six rows high with the necessary blocks on top, proceeding from left to right. Again, however, we have the multiple of 4 restriction, and we carry a surplus of blocks as *state* of the rectangles. This state is visible as indent of up to three blocks at the bottom in the left column of the rectangle. The last rectangle is designed to fill both the final block of the platform and the six rows of the rectangles, clearing all additional blocks that are not part of the final configuration.



As always, the number of blocks occupied in the construction need not be a multiple of 4, and we have to take this into account. We solve this by allowing a group of up to three additional blocks placed on top of one of the blocks. This *overflow* is indicated by + in the figure to the right. The overflow is used as a starting point in the construction of the platform for the next row of blocks. If there is no overflow then we start the construction by putting a horizontal I on top of one of the blocks of the last row (artificially introducing an overflow of four). In the case of odd width, the overflow can be removed each time.



The precise form of the rectangles (for each state, number of blocks and overflow) is rather tedious, see [14]. Particular care has to be taken for the last rectangle which has to clear the intermediate rows.

The whole construction starts with a horizontal I. It is extended to a platform with state 1. In order to remove the last overflow — if any —, the construction ends with some simple final details.

Note that in many cases there exist simpler constructions (for instance for boards of odd width), but the uniform approach has its own merits. Indeed, some configurations are even extremely simple to reach (e.g., a single vertical I), whereas our construction uses an abundance of pieces, clearing many rows on the way.

6 Conclusion

So far we have discussed several issues that are somehow attached to the game of TETRIS. The fact that a well-known and easy to understand game as TETRIS possesses such a rich structure is really surprising. There clearly is a connection between deep mathematical ideas such as NP-completeness and every day life. This fits in the larger research picture, where for many games these sorts of problems are addressed, cf. [1, 6].

Many problems remain open. Among others, one can think of variants of the game rules, but also of more generals topics as the characterization of the clearing sequences in decidable cases.

References

[1] E.R. Berlekamp, J.H. Conway, and R.K. Guy. *Winning Ways for Your Mathematical Plays.* Volume 1 and 2, Academic Press, New York, 1982.

[2] R. Breukelaar, E.D. Demaine, S. Hohenberger, H.J. Hoogeboom, W.A. Kosters, and D. Liben-Nowell. Tetris is Hard, Even to Approximate. International Journal of Computational Geometry and Applications (IJCGA) 14 (2004) 41–68.

[3] R. Breukelaar, H.J. Hoogeboom and W.A. Kosters. Tetris is Hard, Made Easy. Technical Report 2003–9, Leiden Institute of Advanced Computer Science, 2003.

[4] J. Brzustowski. *Can You Win at Tetris?* Master's Thesis, University of British Columbia, 1992.

[5] H. Burgiel. How to Lose at Tetris. Mathematical Gazette 81 (1997) 194–200.

[6] E.D. Demaine. Playing Games with Algorithms: Algorithmic Combinatorial Game Theory. Proceedings 26th International Symposium on Mathematical Foundations of Computer Science MFCS2001, J. Sgall, A. Pultr and P. Kolman (editors), Lecture Notes in Computer Science 2136, pages 18–32, 2001.

[7] E.D. Demaine, S. Hohenberger and D. Liben-Nowell. Tetris is Hard, Even to Approximate. Computing and Combinatorics, 9th Annual International Conference, T. Warnow and B. Zhu (editors), Lecture Notes in Computer Science 2697, pages 351–363, 2003.

[8] B. Durand and L. Vuillon (editors). *Tilings of the Plane.* Special issue of Theoretical Computer Science, Volume 303, Numbers 2–3, 2003.

[9] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, New York, 1979.

[10] S. Ginsburg. *The Mathematical Theory of Context-Free Languages.* McGraw-Hill, New York, 1966.

[11] S.W. Golomb. *Polyominoes: Puzzles, Patterns, Problems, and Packings.* 2nd ed., Princeton University Press, Princeton, NJ, 1995.

[12] H.J. Hoogeboom and W.A. Kosters. Website — How to Construct Tetris Configurations. `http://www.liacs.nl/home/kosters/tetris/`.

[13] H.J. Hoogeboom and W.A. Kosters. Tetris and Decidability. Information Processing Letters 89 (2004) 267–272.

[14] H.J. Hoogeboom and W.A. Kosters. How to Construct Tetris Configurations. International Journal of Intelligent Games & Simulation (IJIGS) 3 (2004) 94–102.

[15] E.L. Post. A Variant of a Recursively Unsolvable Problem. Bulletin of the American Mathematical Society 52 (1946) 264–268.

[16] D.W. Walkup. Covering a Rectangle with T-Tetrominoes. American Mathematical Monthly 72 (1965) 986–988.

The Joys of Graph Transformation

Arend Rensink
Department of Computer Science, University of Twente
P.O.Box 217, 7500 AE, The Netherlands
rensink@cs.utwente.nl

February 10, 2005

Abstract

We believe that the technique of graph transformation offers a very natural way to specify semantics for languages that have dynamic allocation and linking structure; for instance, object-oriented programming languages, but also languages for mobility. In this note we expose, on a rather informal level, the reasons for this belief. Our hope in doing this is to raise interest in this technique and so generate more interest in the fascinating possibilities and open questions of this area.

1 Graph Transformation Is Easy

Transformation means changing (literally: shaping) one thing into another. In the case of graph transformation, obviously, the things being changed are graphs. A fundamental assumption in studying such changes is that they are not arbitrary but controlled by some guiding principles, and that these principles can be captured in *rules*. A graph transformation rule (often called a production rule) describes a *kind* of change that will transform certain graphs — those to which the rule is applicable — into others, in a specific way encoded in the rule. A set of production rules is usually called a production system; graphs that are subjected to transformation are often called host graphs.

For us, the interest in this arises from the fact that graphs can be used to model just about any discrete structure — with lesser or greater ease — and that many kinds of dynamic changes in such structures lend themselves quite naturally to a description by graph production rules. This is in particular true of the semantics of object-oriented systems: it is our firm belief that graph transformations are a very natural technique to specify the semantics of such systems (see also Sect. 4). (It should be mentioned that this is not the only, and indeed historically not the original, reason to be interested in graph transformation: another motivation is to characterise graph structures as the end product of a sequence of transformations guided by a given set of rules. As a very simple example, the class of all connected graphs can be characterised in this way. In that context it is not so much the process of change as its result that is of primary interest.)

Although there are many ways to define graph production rules, all rules have certain basic things in common. They always specify changes in a (relatively) small sub-structure, and the change always consists of modifications to that sub-structure, such as taking away parts from it or adding parts to it. For the rule to apply, the first requirement is that the host graph actually contains a sub-structure of the right kind; in fact, if it contains more than one such sub-structure, the rule is applicable in different ways.

This is, of course, a very general description; in practice, there have turned out to be many different useful ways to specify sub-structures and changes. In the past this has led to strong opinions about the relative merits of the various techniques. Fortunately, the purpose of this paper is not to categorise these approaches (for that, the handbook [21] is a

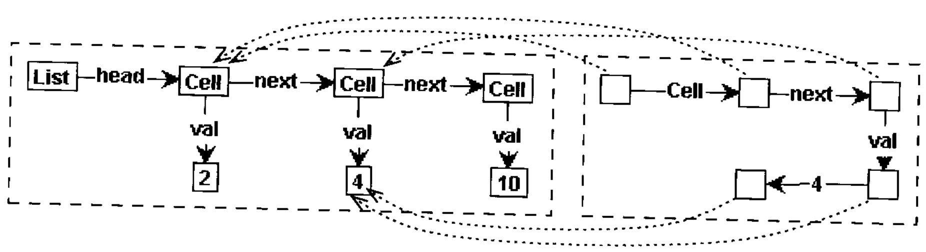


Figure 1: Two graphs with a matching

good source); rather, we want to illustrate the ease with which graph transformation can be used and understood. For this purpose we will concentrate on one, very simple, formalism, which we do not claim to be superior in any formal way but which we do believe to be easily understandable. The approach has been implemented in a tool, called GROOVE, which can be downloaded at `http://groove.sf.net` (see also [17]); all examples in this paper have been produced using the tool.

We will now give some formal definitions. First of all we define the concept of a graph. For this purpose we assume a known set of labels Labels, which contains names with which we will label the edges of our graphs.

Definition 1 (graph) *A graph over* Labels *is a tuple* $\langle \mathsf{Nodes}, \mathsf{Edges}, \mathsf{src}, \mathsf{tgt}, \mathsf{lab} \rangle$, *where* Nodes *and* $\mathsf{Edges} \subseteq \mathsf{Nodes} \times \mathsf{Labels} \times \mathsf{Edges}$ *are finite sets.*

Thus, an edge $\mathsf{e} \in \mathsf{Edges}$ is a triple $(\mathsf{v}, \mathsf{a}, \mathsf{w})$ consisting of a source node $\mathsf{src(e)} = \mathsf{v}$, a label $\mathsf{lab(e)} = \mathsf{a}$, and a target node $\mathsf{tgt(e)} = \mathsf{w}$. If $\mathsf{src(e)} = \mathsf{tgt(e)}$ we call e a *self-edge*. If G is a graph then we use $\mathsf{G.Nodes}$ and $\mathsf{G.Edges}$ to indicate its components. Note that the definition rules does not allow different edges with the same source nodes, labels and target nodes. If $\mathsf{G.Nodes} \subseteq \mathsf{H.Nodes}$ and $\mathsf{G.Edges} \subseteq \mathsf{H.Edges}$, we call G a *subgraph* of H; we indicate this by $\mathsf{G} \subseteq \mathsf{H}$. We now define what it means for one graph to match another.

Definition 2 (graph matching) *Let* G, H *be graphs. A matching of* G *in* H *is given by a function* $\mathsf{nodeMap}\colon \mathsf{G.Nodes} \to \mathsf{H.Nodes}$, *such that*

$$(\mathsf{v}, \mathsf{a}, \mathsf{w}) \in \mathsf{G.Edges} \textit{ implies } (\mathsf{nodeMap(v)}, \mathsf{a}, \mathsf{nodeMap(w)}) \in \mathsf{H.Edges} \ .$$

We write $\mathsf{nodeMap}\colon \mathsf{G} \to \mathsf{H}$ to denote that $\mathsf{nodeMap}$ is a matching of G in H. We also extend $\mathsf{nodeMap}$ pointwise to edges; i.e.,

$$\mathsf{nodeMap(e)} = (\mathsf{nodeMap(src(e))}, \mathsf{lab(e)}, \mathsf{nodeMap(tgt(e))}) \ .$$

Example 3 *Fig. 1 shows two graphs, with the notational convention that labels shown on nodes are shorthand for self-edges with those labels. The left hand graph shows a linked-list structure, consisting of* Cell*-labelled nodes (or, more accurately, nodes with* Cell*-labelled self-edges) linked by* next*-labelled edges. The right hand side graph shows another graph over the same label set. Furthermore, the dotted arrows indicate the node map of a matching of the right hand side graph in the left hand side graph. The edge map is not depicted but can be constructed uniquely from the node map (again taking into account that the node labels* Cell *and* $\mathsf{4}$ *on the left hand side actually stand for self-edges).*

Definition 4 (production rule) *A production rule is a tuple* $\langle \mathsf{Lhs}, \mathsf{Rhs} \rangle$ *of graphs. The rule is* applicable *to a graph* G *if there is a matching of* Lhs *into* G.

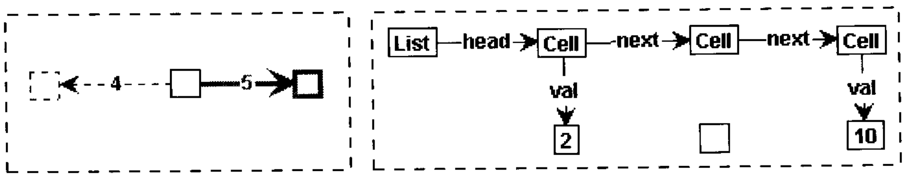


Figure 2: A production rule, and the result of its application to Fig. 1

To define the application of a rule $\langle \mathsf{Lhs}, \mathsf{Rhs} \rangle$ precisely, we identify the sets of elements scheduled to be deleted and those scheduled to be created, as follows:

$$\begin{aligned}
\mathsf{Del.Nodes} &= \mathsf{Lhs.Nodes} \setminus \mathsf{Rhs.Nodes} \\
\mathsf{Del.Edges} &= \mathsf{Lhs.Edges} \setminus \mathsf{Rhs.Edges} \\
\mathsf{New.Nodes} &= \mathsf{Rhs.Nodes} \setminus \mathsf{Lhs.Nodes} \\
\mathsf{New.Edges} &= \mathsf{Rhs.Edges} \setminus \mathsf{Lhs.Edges} .
\end{aligned}$$

When we apply a rule to G, the elements that are in Lhs but not in Rhs are deleted from G (or rather, their images under the matching) and elements that are in Rhs but not in Lhs are added (or rather, fresh images are created for them). However, things are complicated by two effects:

- The matching may be non-injective; in particular, there may be $\mathsf{v}_1 \in \mathsf{Del.Nodes}$ and $\mathsf{v}_2 \in \mathsf{Lhs.Nodes} \setminus \mathsf{Del.Nodes}$ such that $\mathsf{nodeMap}(\mathsf{v}_1) = \mathsf{nodeMap}(\mathsf{v}_2)$;
- The matching may be non-surjective on the incident edges of a node scheduled to be deleted; in particular, there may be $v \in \mathsf{Del.Nodes}$ and $\mathsf{e} \in \mathsf{G.Edges} \setminus \mathsf{nodeMap}(\mathsf{Del.Edges})$ such that $\mathsf{G.src(e)} = \mathsf{nodeMap(v)}$ or $\mathsf{G.tgt(e)} = \mathsf{nodeMap(v)}$. (Such an edge e is often called a *dangling edge.*)

These problems are resolved by specifying that deletion always wins out: that is, $\mathsf{nodeMap}(\mathsf{v}_1)$ and e in the scenarios above will both be deleted from G. This can have unexpected effects, as the example below will show.

Example 5 *Fig. 2 shows a graph production rule, where the left hand side and right hand side are combined into one graph, with the following notational conventions:*

- Del.Nodes *and* Del.Edges *are depicted as thin, dashed (blue) nodes and edges;*
- New.Nodes *and* New.Edges *are depicted as wide (green) nodes and edges;*
- *All other nodes and edges are in the intersection of* Lhs *and* Rhs.

In other words, Lhs *consists of the thin (continuous and dashed) nodes and edges and* Rhs *consists of the continuous (thin and wide) nodes and edges.*

Intuitively, the rule in Fig. 2 specifies that a 4*-labelled edge is to be deleted, together with its target node, and a* 5*-labelled edge is to be created to a new node. However, the only matching of* Lhs *in the left hand graph of Fig. 1 maps both rule nodes to the same graph node; moreover, this graph node also has an incoming* val*-edge, about which the rule says nothing. The result is that the node is deleted, with its incoming* val*-edge; even more curiously, the* 5*-edge that was just created is deleted as well, but the node that was also just created is preserved. The result is also shown in Fig. 2.*

The following algorithm defines how to apply a graph production rule $\langle \mathsf{Lhs}, \mathsf{Rhs} \rangle$ to a host graph G, given a matching nodeMap. The resulting transformed graph is denoted H.

1. Extend nodeMap to a total function $\mathsf{nodeMap}_1$ from $\mathsf{Lhs.Nodes} \cup \mathsf{Rhs.Nodes}$ by adding fresh images (not in G.Nodes) for all elements of New.Nodes;
2. Construct a graph $\mathsf{K} = (\mathsf{nodeMap}_1(\mathsf{Rhs.Nodes}), \mathsf{nodeMap}_1(\mathsf{Rhs.Edges}))$;

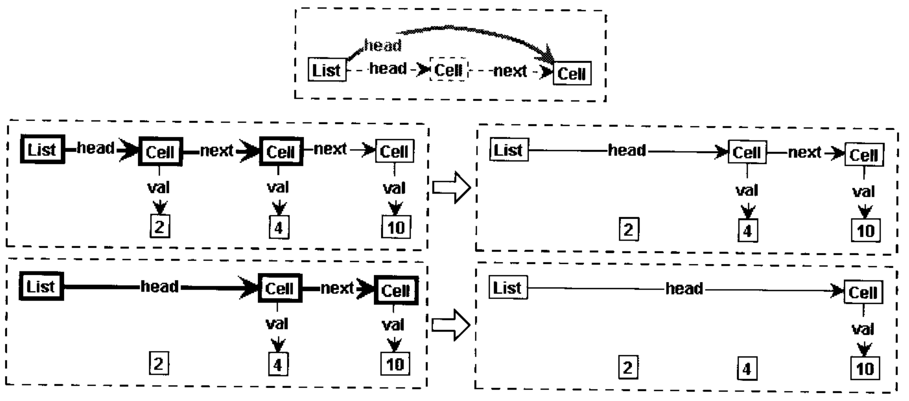


Figure 3: Rule for deleting the head element of a list, with two applications

3. Construct H such that

$$\begin{aligned}\mathsf{H.Nodes} &= \mathsf{K.Nodes} \setminus \mathsf{nodeMap(Del.Nodes)}\\ \mathsf{H.Edges} &= (\mathsf{K.Edges} \setminus \mathsf{nodeMap(Del.Edges)})\\ &\quad \cap\, \mathsf{K.src}^{-1}(\mathsf{H.Nodes}) \cap \mathsf{K.tgt}^{-1}(\mathsf{H.Nodes})\ .\end{aligned}$$

It should be noted that the complications identified above (conflicts between preservation and deletion, and dangling edges) may also be resolved in a different manner, namely by strengthening the conditions under which a rule is considered to be applicable (see Def. 4) so that these cases are automatically excluded. In fact, the solution presented above is the one followed in the so-called *single-pushout approach* whereas the *double-pushout approach* strengthens the application condition instead — see [6, 12] for a thorough discussion. The latter has the advantage that the graphs K and H in the construction above always coincide; in other words, the construction of the transformed graph becomes easier.

Example 6 *Fig. 3 shows a more useful rule than the one in Fig. 2: it deletes the first cell in a list, provided this is not the last one. The figure also shows an application of this rule to the list graph of Fig. 1. On the left, the image of the matching in the host graph is emphasised. In this case, since* New.Nodes *is empty, the extended mapping* $\mathsf{nodeMap}_1$ *used in the construction above equals* nodeMap*; consequently, the* K *in the construction equals the host graph with one additional* head*-edge. To obtain the target graph, the first* Cell*-node with its incident edges (including the dangling* val*-edge, which is not explicitly mentioned in the rule) is removed from* K*. In the resulting graph, the same rule is applicable once more, resulting in the second transformation in the figure. In the final graph, now, the rule is no longer applicable since no matching from the left hand side exists: the first cell in the list no longer has a successor. Below (Ex. 10) we will show how to define a second rule that takes care of removing the last remaining cell, while making sure that this is only applicable when appropriate.*

2 Graph Transformation Is Logical

The existence of a matching of a graph G in another graph H can be interpreted as a condition on H, namely that the structure described by G can be found somewhere in it. This is for instance used in the applicability condition we have defined in Def. 4: in order to apply a rule, a matching from its left hand side in the host graph must exist. Unfortunately, the kind of properties we can express in this way is limited: essentially, they can only state the *existence* of nodes and edges.

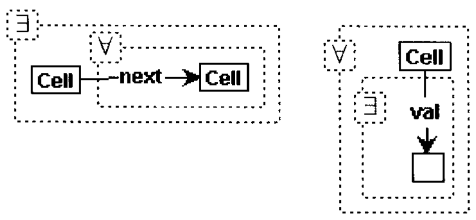


Figure 4: The graph properties of Ex. 7 as nested graphs

Example 7 *Here are some examples of useful properties that are not existential, in the sense above:*

- *Negative conditions: for instance, the property that a given* Cell*-node does* not *have an outgoing* next*-edge. This would be useful to define a rule, complementary to the one in Fig. 3, that deletes the only remaining* Cell*-node from a list.*
- *Invariants: for instance, the property that all* Cell*-nodes have an outgoing* val*-edge. This is a typical property that one would want to verify on a given system.*

A much more powerful way of specifying graph properties, still based on the same principle of graph matching, is obtained by *nesting* the graphs to be matched inside each other and alternating existential and universal quantors for each level of nesting. Two examples, expressing precisely the properties of Ex. 7, are shown in Fig. 4 below.[1]

Symbolically, such nested structures can be understood as existential or universal formulae, exist or univ, respectively, in the following grammar (I and J are finite sets and P, Q_i for $i \in I$ and R_j for $j \in J$ are "property graphs"):

$$\begin{aligned} \mathsf{exist}(\mathsf{P}) \quad &::= \quad \textstyle\bigvee_{i\in I} \exists \mathsf{Q}_i\ \mathsf{univ}_i(\mathsf{Q}_i) \quad (\mathsf{P} \subseteq \mathsf{Q}_i) \\ \mathsf{univ}(\mathsf{P}) \quad &::= \quad \textstyle\bigwedge_{j\in J} \forall \mathsf{R}_j\ \mathsf{exist}_j(\mathsf{R}_j) \quad (\mathsf{P} \subseteq \mathsf{R}_j)\ . \end{aligned}$$

This grammar should be read as follows. Each formula form (either exist or univ) is parameterised with a property graph P, which corresponds to the structure that has already been matched in some outer formula (i.e., within which form is a sub-formula). Each next level quantifies over a super-graph of P, meaning that it requires the existence of additional structure in the host graph (in the case of exist) or states a universal property over all instances of such additional structure (in the case of univ). The semantics is defined by the following rules, where $\mathsf{m}\colon \mathsf{P} \to \mathsf{G}$ stands for the matching already established for the parameter graph P in the host graph G:

$$\begin{aligned} \mathsf{m} \models \mathsf{exist}(\mathsf{P}) \ &:\Leftrightarrow \ \text{there are } i \in I, \mathsf{n}\colon \mathsf{Q}_i \to \mathsf{G} \text{ such that } \mathsf{m} \subseteq \mathsf{n} \text{ and } \mathsf{n} \models \mathsf{univ}_i(\mathsf{Q}_i) \\ \mathsf{m} \models \mathsf{univ}(\mathsf{P}) \ &:\Leftrightarrow \ \text{for all } j \in J, \mathsf{n}\colon \mathsf{R}_j \to \mathsf{G}, \text{ if } \mathsf{m} \subseteq \mathsf{n} \text{ then } \mathsf{n} \models \mathsf{exist}_j(\mathsf{R}_j)\ . \end{aligned}$$

(The notation $\mathsf{m} \subseteq \mathsf{n}$, where $\mathsf{m}\colon \mathsf{P} \to \mathsf{G}$ and $\mathsf{n}\colon \mathsf{Q} \to \mathsf{G}$ with $\mathsf{P} \subseteq \mathsf{Q}$, means that we can create n from m by adding images (in G) for the nodes in Q.Nodes$\setminus$P.Nodes.) It should be remarked that the sets I and J may be empty: $\bigvee \emptyset$ is logically equivalent to **ff** and $\bigwedge \emptyset$ to **tt**. As a consequence, a sub-formula $\forall \mathsf{R}_j\ \mathsf{exist}_j(\mathsf{R}_j)$ where $\mathsf{exist}_j = \bigvee \emptyset$ actually expresses the *non-existence* of the structure R_j in the host graph. An example of this is found in the left hand side condition of Fig. 4: the inner, ∀-quantified structure is in fact forbidden.

The rules above define under what circumstances a formula is satisfied by a *matching*. In the end, however, we want to use such formulae to specify properties of *graphs*. For this purpose, we use the one-to-one correspondence between graphs and matchings of the *empty property graph*, here denoted $\emptyset$, into it: that is, we equate a host graph G with (the

[1]This representation as nested graphs is essentially that of *existential graphs* as studied in, e.g., [20, 26, 7]. We have worked out an alternative, more elegant but much more complicated, representation as tree-shaped diagrams in the category of graphs in [18].

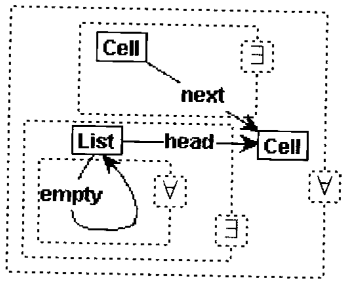


Figure 5: Nested graph property with an existential disjunction

unique) matching m: $\emptyset \rightarrow$G. To be precise, we define, for arbitrary formulae form($\emptyset$) (either exist($\emptyset$) or univ($\emptyset$)):

$$\mathsf{G} \models \mathsf{form} \quad :\Leftrightarrow \quad \mathsf{m}\colon \emptyset \rightarrow \mathsf{G} \models \mathsf{form}\ .$$

Example 8 *Another, more involved example of a nested graph property is given in Fig. 5. This expresses that every cell either has a predecessor or is the head element of a list, in which case, moreover, the list does not have an* empty*-self-edge. The graph of Fig. 1 satisfies this property: it allows three matchings of the outermost (universally quantified) graph, one of which can be extended to a matching of the bottom (existentially quantified) inner graph, where, moreover, the innermost (universally quantified) graph, consisting of the* empty*-edge, is absent; the other two outermost matchings can be extended to matchings of the top (existentially quantified) inner graph.*

Clearly, with these nested graphs we can express much stronger properties than with single matchings. The following is a consequence of the work on existential graphs, and also follows from [18]:

Theorem 9 *Nested graphs, interpreted as graph properties in the way defined above, are as expressive as first order logic with binary relations (but without equality).*

The restriction to logic without equality is actually quite unfortunate, since it prevents us from doing any kind of *counting*. There are many useful properties that involve, for instance, the uniqueness of nodes with a certain property; for instance, the fact that (in our list example) Cell-nodes are unshared, i.e., have at most one predecessor. It follows from Th. 9 that this and similar properties cannot be expressed with the nested graphs presented above. There are several possible ways to lift this restriction.

- Limit the matchings allowed in Def. 4 to injective ones only. This is a definition often seen in the graph transformation literature: although it may cost a (in the worst case super-exponential) blowup in the number of rules required to model a particular behaviour, in many practical examples it seems quite reasonable.
- Extend the subgraph relation between P and the Q_i and R_j in the grammar of exist and univ to (non-injective) matchings $\mathsf{m}_i\colon \mathsf{P} \rightarrow \mathsf{Q}_i$ resp. $\mathsf{n}_j\colon \mathsf{P} \rightarrow \mathsf{R}_j$. This is the solution presented in [18], at the cost of additional technical complexity. A consequence is that the resulting structure cannot be depicted as a nested graph any more.
- Introduce special edges in the property graphs that stand for the equality of nodes. We present this solution below.

Negative application conditions. The idea of using some form of nesting to enhance the control over rule applications is far from new: it was described first in [14] for a single level of nesting (resulting in so-called negative application conditions) and extended in [15]

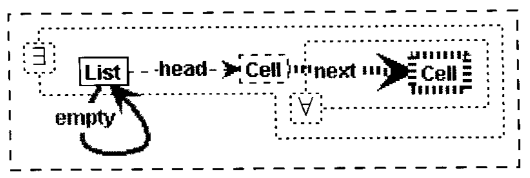


Figure 6: Deletion of the last remaining element from a list

to a second level of nesting (so-called conditional application conditions). The idea is to regard the LHS Lhs of a rule ⟨Lhs, Rhs⟩ as the starting point of a formula ∃Lhs univ(Lhs). Any matching that is discovered to satisfy the sub-formula univ(Lhs) gives rise to a rule application.

Example 10 *Fig. 6 shows a production rule for deleting the head element of a list in case this is also the final element, i.e., if it has no outgoing* next*-edge. This involves a negative application condition, here depicted as a nested universal sub-graph (without further sub-conditions, so that the universal property works as a negative condition). As a consequence, this rule is not applicable in the list graph of Fig. 1, but it is applicable to the final (bottom right hand) graph of Fig. 3.*

In addition to the explicit bounding box for the negative condition, in Fig. 6 we have also distinguished the relevant subgraph by using very wide, closely dashed (red) lines. In future examples of rules with negative application conditions we will actually leave out the bounding boxes: each connected sub-graph drawn in these wide, closely dashed lines implicitly stands for a universal sub-formula without further sub-formulae, which is therefore works as a negative condition.

Equality and regular expressions. So far we have used ordinary graph morphisms as matchings. We now strengthen the type of properties we can express by using a different class of labels in P, which should not be matched by single edges of G but rather by (possibly empty) paths through G. We will use H.Paths to denote the set of such paths. Formally:

$$\mathsf{H.Paths} = \{\mathsf{v}_1{\cdot}\mathsf{a}_1{\cdot}\mathsf{v}_2\cdots\mathsf{a}_{n-1}{\cdot}\mathsf{v}_n \in \mathsf{H.Nodes}{\cdot}(\mathsf{Labels}{\cdot}\mathsf{H.Nodes})^* \mid \\ \forall 1 \le i < n : (\mathsf{v}_i, \mathsf{a}_i, \mathsf{v}_{i+1}) \in \mathsf{H.Edges}\} \ .$$

If p ∈ G.Paths for some graph G, we use lab(p) to stand for the sequence of labels in p, src(p) for the first node and tgt(p) for the last node in p; hence, if $\mathsf{p} = \mathsf{v}_1{\cdot}\mathsf{a}_1{\cdot}\mathsf{v}_2\cdots\mathsf{a}_{n-1}{\cdot}\mathsf{v}_n$ then $\mathsf{src(p)} = \mathsf{v}_1$, $\mathsf{lab(p)} = \mathsf{a}_1\cdots\mathsf{a}_{n-1}$ and $\mathsf{tgt(p)} = \mathsf{v}_n$.

Definition 11 (path expression language) *A* path expression language *over* Labels *is a pair* ⟨Exprs, sat⟩, *where* Exprs *is a set of path expressions with* Labels ⊆ Exprs, *and* sat⊆ Labels* × Exprs *a* satisfaction relation *between sequences of labels in* Labels *and path expressions, such that for all* a ∈ Labels*:*

$$\mathsf{s\ sat\ a} \iff \mathsf{s} = \mathsf{a} \ .$$

If (s, x) ∈sat for some label sequence s ∈ Labels* and path expression x ∈ Exprs we say that s *satisfies* x; we usually denote this in infix notation, as s sat x. The definition specifies that a path expression that is actually an element of Labels (recall that Labels ⊆ Exprs) is satisfied only by itself. A prime example of a path expression language is that of *regular expressions*, RegExprs, generated by the following grammar:

$$\mathsf{x} ::= \ = \mid \mathsf{a} \mid \mathsf{x}|\mathsf{x} \mid \mathsf{x}\cdot\mathsf{x} \mid \mathsf{x}^* \ .$$

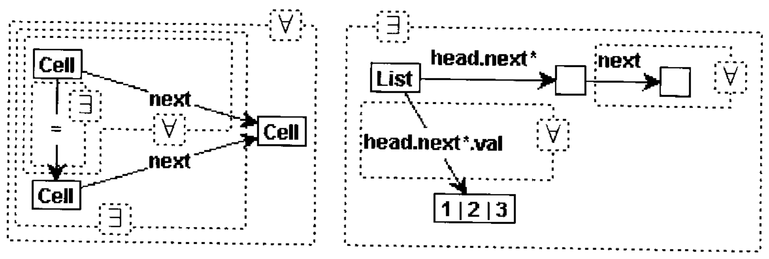


Figure 7: Nested graph properties with path expressions

(where $=$ stands for the empty sequence.) Satisfaction is defined as usual:

$$\begin{array}{rcl}
\mathsf{s\ sat} = & :\Leftrightarrow & \mathsf{s} = \varepsilon \\
\mathsf{s\ sat\ a} & :\Leftrightarrow & \mathsf{s} = \mathsf{a} \\
\mathsf{s\ sat\ x_1|x_2} & :\Leftrightarrow & \mathsf{s_1\ sat\ x_1\ or\ s_2\ sat\ x_2} \\
\mathsf{s\ sat\ x_1 \cdot x_2} & :\Leftrightarrow & \exists \mathsf{s_1, s_2} \in \mathsf{Labels}^* : \mathsf{s} = \mathsf{s_1 \cdot s_2}, \mathsf{s_1\ sat\ x_1}, \mathsf{s_2\ sat\ x_2} \\
\mathsf{s\ sat\ x^*} & :\Leftrightarrow & \exists \mathsf{s_1}, \ldots, \mathsf{s}_n \in \mathsf{Labels}^* : \mathsf{s} = \mathsf{s_1} \cdots \mathsf{s}_n, \forall 1 \le i \le n : \mathsf{s}_i\ \mathsf{sat\ x} \ .
\end{array}$$

For instance, both ε sat next*, next sat next* and next·next sat @—*, and both 1 sat 1|2|3 and 2 sat 1|2|3. We generalise the notion of matching to graphs over path expressions as follows:

Definition 12 (path matching) *Given a path expression language* ⟨Exprs, sat⟩ *over* Labels*, a path matching of a graph* P *over* Exprs *(i.e., where the labels in* P *are actually path expressions) into a graph* G *over* Labels *is a function* nodeMap: P.Nodes → G.Nodes *such that for all* (v, x, w) ∈ P.Edges*:*

$$\exists \mathsf{p} \in \mathsf{G.Paths} : \mathsf{src(p)} = \mathsf{nodeMap(v)}, \mathsf{lab(p)\ sat\ x}, \mathsf{tgt(p)} = \mathsf{nodeMap(w)} \ .$$

Example 13 *Fig. 7 shows two nested graph properties. The property on the left hand side is an invariant, expressing that every* Cell*-node has a unique predecessor. Note that the* $=$*-labelled edge specifies that its source and target node are matched by the same node of the host graph, i.e., that the matching is non-injective on these nodes.*

The right hand side is an existential property, expressing that from a List*-node there is a* Cell*-node without successor, and moreover, that no reachable cell node has a* val*-edge to a node with self-edge 1, 2 or 3. For instance, in the initial (top left hand) graph of Fig. 3, this property is not fulfilled because the 2-labelled node still connected to the list; but once we have deleted the first* Cell*-node (bottom left) the property holds, as indicated by the emphasised part of the graph in Fig. 7.*

It should be clear from this example that path expressions enhance the expressive power of nested graph properties. On the one hand we can now state properties about node equality, using $=$-labelled edges; hence we have gained (among other things) the ability to count nodes. On the other hand, the Kleene star enables us to specify paths of arbitrary length, which implies that we can state properties about transitive closure and connectivity; this takes us outside standard first-order logic. Thus, we have the following result (compare Th. 9):

Theorem 14 *Nested graphs with path expressions, interpreted as graph properties in the way defined above, are properly more expressive than first order logic with binary relations and equality.*

In order to use this expressive power in graph transformations, we extend the definition of a rule (Def. 4) so that Lhs and Rhs are graphs over Exprs rather than Labels. However, this only makes sense for that part of the rule that controls the applicability: it is not clear

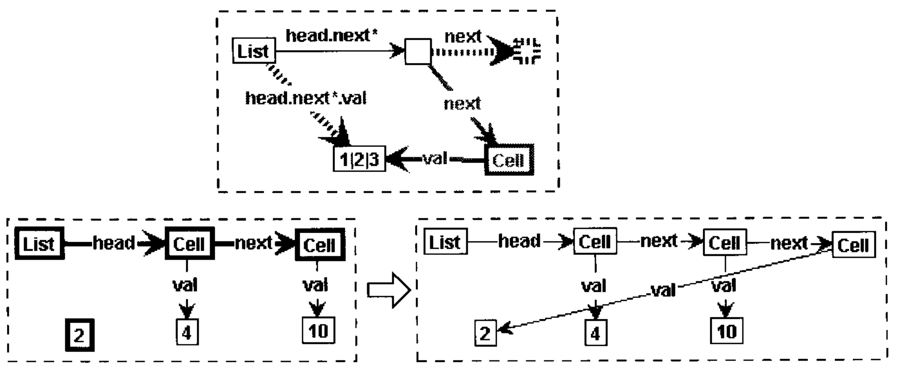


Figure 8: Production rule with path expressions and an application

what it would mean if Del.*Edg* or New.Edges are Exprs-labelled. Instead, for the edges scheduled to be added or deleted we want to use

Definition 15 (production rule with path expressions) *A production rule with path expressions is a tuple* ⟨Lhs, Rhs⟩ *of graphs over* Exprs, *where* lab(e) ∈ Labels *for all* e ∈ Del.Edges ∪ New.Edges. *The rule is* applicable *to a graph* G *if there is a path matching of* Lhs *into* G.

The application of a production rule with path expressions is the same as for ordinary production rules, thanks to the fact that the deleted and created nodes and edges take their labels from Labels.

Example 16 *Fig. 8 shows a production rule, essentially consisting of the nested graph property of Fig. 7 as its left hand side. The right hand side specifies that the unconnected node identified in the LHS should be appended to the list.*

3 Graph Transformation Is Difficult

We have advertised graph transformation as a technique that is easy to grasp, powerful and (in certain application domains) natural. Why, then, is this technique not part of the tool kit of more formal methods researchers?

We believe that part of the answer to this question is: because the area of graph transformation is perceived to be difficult, and to be centred on questions of a very theoretical nature. And there is certainly some justification for this feeling. A sizeable fragment of the literature on graph transformation treats the subject on a very abstract level, in a categorical setting; for those not familiar or even averse to category theory, this can be a great barrier to appreciating the joys of graph transformation. In this perspective, it is almost certainly a mistake in public relations to use terms like "Single Pushout Approach" to refer to an intuitively straightforward technique: it suggests that, in order to apply the technique, one has to know what a pushout is, whereas we believe that this is completely irrelevant in most cases.

This is not to deny that category theory is a marvellous way to abstract away from differences in the particular kinds of graphs that one is transforming (typed or untyped, with binary or hyperedges, flat or hierarchical, with or without attributes, to name some choices), and there is much to be gained from lifting some of the general issues involved in any kind of transformation (or rewriting) to this abstract level. For those interested in such theoretical issues we briefly discuss some of them.

Independence of transformations (see, e.g., [6, 11]). This refers to the question if two transformations have overlapping or conflicting effects — for instance, because one transformation removes a node or edge that the other requires in order to be applicable. If such conflicts do not occur, the transformations are called *independent*. This may have important consequences: For instance, in the context of system verification, it may be unnecessary to compute all transformations even in a full state space exploration. In *op. cit.* this problem is studied on the level of properties of the categories involved, so that it does not have to be done again for all different types of graphs.

Constraints and conditions on graphs and graph transformations (sse, e.g., [14, 15, 9, 24]). In Sect. 2 we have shown a fairly straightforward way to formulate properties of our graphs, but again, one would prefer not to re-invent the wheel for each particular type of graph. There are two places where properties of graphs play a special role in the framework of graph transformation: as *invariants* that one would like to hold on all graphs, or as *application conditions* that control the applicability of rules. In *op. cit.* these types of properties and their interrelation are studied.

Compositionality of graph production systems (see, e.g., [13]). In process algebra terminology, graph transformations define a *reduction semantics*: each state "reduces" to the next without interference from the environment — in other words, graph transformations define a "closed world semantics" in which all components of the system being modelled have to be included in the model. In *op. cit.* it is studied how the transformation framework can be enriched with contextual information, so that the behaviour of a complete system can instead be modelled on the level of individual components, which can be put together afterwards.

4 Graph Transformation Wants You!

Although the field of graph transformation stems from the beginning of the 70s and so is, to computer science standards, downright venerable, its application to behavioural modelling and in particular behavioural verification is relatively young. In this contribution we hope to have given you a taste of how it works and an impression of how it could be applied in that area, but there are many open issues. We list a few of them here, with some references; we believe all of these areas to be worthy of further investigation.

Specification. In Sect. 1 we have called graph transformations a natural technique for the specification of semantics for dynamic changes, on particular in object-oriented systems. One paper in which this has been worked out in concrete detail for a relatively large fragment of Java is [5]; another approach is taken in [8], who (essentially) define a special object-based language with a graph transformation semantics. However, we are yet far removed from a discipline in which the definition of such a semantics can be easily and systematically written down — essentially an EBNF standard for behavioural semantics.

Verification. Starting with a graph production system, one can generate the corresponding transition system, essentially by computing all rule applications recursively. This opens up the way for extending existing methods for test generation or model checking to graph transformation-based systems. Some studies can be found in [8, 19, 25]; there are, however, major open problems in dealing with the dynamic nature of the states (which cannot be captured by a fixed state vector). Another interesting option is to extend assertional reasoning to graph production rules: the theory of graph properties mentioned in Sect. 3 lays down at least the terminology in which this problem may be addressed, but the connection to predicate transformation yet has to be tackled. Yet another option is to regard a graph production system as an extended Petri net and transfer techniques from that area, as seen, e.g., in [1, 2].

Abstraction. In the context of the verification issue, we would especially like to address abstraction, as an approximative technique. We believe that graphs offer a very clean setting to study state abstractions; this brings us in the realm of static *shape* analysis á la Sagiv et al. [22, 23] and abstract interpretation. A first proposal was described in [16]. Abstraction of a different kind, using principles from Petri net unfoldings, have also been proposed in [3, 4].

Compositionality. In Sect. 3 we have briefly referred to the general theory for compositionality that has recently been developed. However, this work takes the reduction semantics as a starting point and derives contextual rules from that. We believe that, instead, it can be quite natural to start off with transformation rules that explicitly take context into account, and whose effect may include the sending and receiving of (sub)graphs. For instance, in the context of the running example on lists (e.g., Ex. 16), one may imagine a rule that appends an object "received" from some other component of the system; the identity of the object may be communicated through some parameterisation mechanism in the style of Structural Operational Semantics. As far as we are aware, this combination of ideas from graph transformation and process algebra has not been considered at all so far.

As a consequence of the predominance of category theory in much of the graph literature research, it is also obligatory to place this paper within the categorical framework. What we have defined, on a set-theoretic level, is an instance of the single-pushout approach, which cannot be formulated easily in the double pushout approach — the reason essentially being the fact that we do not allow so-called *parallel edges*, or in other words, that our edges do not have an identity of their own. From the point of view of the algebraic framework, this is a real drawback: much of the theory referred to above applies only in the double pushout approach. The reason why we have nevertheless set up our definitions as we did is twofold: it makes for an easier technical presentation, but more importantly, we like to equate edges to binary relations. In that light it does not make sense for edges to have an identity themselves.

References

[1] P. Baldan, A. Corradini, and B. König. A static analysis technique for graph transformation systems. In K. Larsen and M. Nielsen, editors, *Concur 2001: Concurrency Theory*, volume 2154 of *LNCS*, pages 381–395. Springer-Verlag, 2001.

[2] P. Baldan, A. Corradini, and B. König. Verifying finite-state graph grammars: An unfolding-based approach. In P. Gardner and N. Yoshida, editors, *Concurrency Theory (CONCUR)*, volume 3170 of *LNCS*, pages 83–98. Springer-Verlag, 2004.

[3] P. Baldan, B. König, and B. König. A logic for analyzing abstractions of graph transformation systems. In R. Cousot, editor, *Static Analysis*, volume 2694 of *LNCS*, pages 255–272. Springer-Verlag, 2003.

[4] P. Baldan, B. König, and I. Stürmer. Generating test cases for code generators by unfolding graph transformation systems. In Ehrig et al. [10], pages 194–209.

[5] A. Corradini, F. L. Dotti, L. Foss, and L. Ribeiro. Translating java into graph transformation systems. In Ehrig et al. [10], pages 383–389.

[6] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation, part I: Basic concepts and double pushout approach. In Rozenberg [21], chapter 3, pages 163–246.

[7] F. Dau. *The Logic System of Concept Graphs with Negation*, volume 2892 of *LNCS*. Springer-Verlag, 2003.

[8] F. L. Dotti, L. Foss, L. Ribeiro, and O. M. dos Santos. Verification of distributed object-based systems. In E. Najm, U. Nestmann, and P. Stevens, editors, *Formal Methods for Open Object-based Distributed Systems (FMOODS)*, volume 2884 of *LNCS*, pages 261–275. Springer-Verlag, 2003.

[9] H. Ehrig, K. Ehrig, A. Habel, and K.-H. Pennemann. Constraints and application conditions: From graphs to high-level structures. In Ehrig et al. [10], pages 287–303.

[10] H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors. *Second International Conference on Graph Transformation*, volume 3256 of *LNCS*. Springer-Verlag, 2004.

[11] H. Ehrig, A. Habel, J. Padberg, and U. Prange. Adhesive high-level replacement categories and systems. In Ehrig et al. [10], pages 144–160.

[12] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. Algebraic approaches to graph transformation, part II: Single pushout approach and comparison with double pushout approach. In Rozenberg [21], pages 247–312.

[13] H. Ehrig and B. König. Deriving bisimulation congruences in the dpo approach to graph rewriting. In I. Walukiewicz, editor, *Foundations of Software Science and Computation Structures (FOSSACS)*, volume 2987 of *LNCS*, pages 151–166. Springer-Verlag, 2004.

[14] A. Habel, R. Heckel, and G. Taentzer. Graph grammars with negative application conditions. *Fundamenta Informaticae*, 26(3/4):287–313, 1996.

[15] R. Heckel and A. Wagner. Ensuring consistency of conditional graph grammars – a constructive approach. In A. Corradini and U. Montanari, editors, *Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting and Computation*, volume 2 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers B.V., 1995.

[16] A. Rensink. Canonical graph shapes. In D. A. Schmidt, editor, *Programming Languages and Systems — European Symposium on Programming (ESOP)*, volume 2986 of *LNCS*, pages 401–415. Springer-Verlag, 2004.

[17] A. Rensink. The GROOVE simulator: A tool for state space generation. In J. Pfalz, M. Nagl, and B. Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*, volume 3062 of *LNCS*, pages 479–485. Springer-Verlag, 2004.

[18] A. Rensink. Representing first-order logic using graphs. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *International Conference on Graph Transformations (ICGT)*, volume 3256 of *LNCS*, pages 319–335. Springer-Verlag, 2004.

[19] A. Rensink, Á. Schmidt, and D. Varró. Model checking graph transformations: A comparison of two approaches. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *International Conference on Graph Transformations (ICGT)*, volume 3256 of *LNCS*, pages 226–241. Springer-Verlag, 2004.

[20] D. D. Roberts. The existential graphs. *Computers and Mathematics with Applications*, 6:639–663, 1992.

[21] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume I: Foundations. World Scientific, Singapore, 1997.

[22] M. Sagiv, T. Reps, and R. Wilhelm. Solving shape-analysis problems in languages with destructive updating. *ACM Trans. Prog. Lang. Syst.*, 20(1):1–50, Jan. 1998.

[23] M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Trans. Prog. Lang. Syst.*, 24(3):217–298, May 2002.

[24] G. Taentzer and A. Rensink. Ensuring structural constraints in graph-based models with type inheritance. In *Fundamental Approaches to Software Engineering (FASE)*, LNCS. Springer-Verlag, 2005.

[25] D. Varró. Automated formal verification of visual modeling languages by model checking. *Journal of Software and Systems Modelling*, 3(2):85–113, 2004.

[26] M. Wermelinger. Conceptual graphs and first-order logic. In *International Confence on Conceptual Structures*, volume 954 of *LNAI*, pages 323–337. Springer-Verlag, 1995.

Explicit substitution for graphs

Vincent van Oostrom
Universiteit Utrecht
Faculteit Wijsbegeerte, Theoretische Filosofie
Heidelberglaan 8, 3584 CS Utrecht, The Netherlands
Vincent.vanOostrom@phil.uu.nl

Abstract

We present an atomic decomposition of substitution into erasure, duplication and (for bound variables) scoping.

1. Introduction

Substitution pervades logic. Implementing logic one immediately realises that substitution is not an atomic operation. Thus one is faced with the question how to implement substitution. In the seminal work by De Bruijn on Automath this question was answered by introducing what is now known as an explicit substitution calculus on terms. Here, working on graphs instead of on terms, we present explicit substitution for graphs. In particular, we show how substitution can be made explicit by means of three atomic operators for erasure, duplication, and scoping. Here we aim to present the basic ideas in an intuitive way. We illustrate the issues for rewriting systems, although similar ideas can be found in many other sub-fields of logic.

2. Linear substitution

A term rewriting system (TRS) is given by an alphabet together with a set of rewrite rules over the alphabet [9]. As an example consider the TRS $\mathcal{A}$ for addition of (unary) natural numbers having rules:

$$\begin{aligned} x+0 &\to x \\ x+S(y) &\to S(x+y) \end{aligned}$$

Using these rules we may find the reduction $\mathcal{R}$ given by

$$S(S(0)+S(0)) \to_{\mathcal{A}} S(S(S(0)+0))) \to_{\mathcal{A}} S(S(S(0)))$$

For instance, the first step is obtained by observing that the underlined sub-term of $S(\underline{S(0)+S(0)})$ is a substitution instance of the left-hand side $x+S(y)$ of the second rule (substitute $S(0)$ for x and 0 for y). The step is obtained by replacing this sub-term by taking the same substitution instance for the right-hand side of the same rule, yielding $S(\overline{S(S(0)+0)})$, where the replaced sub-term is over-lined.

The reason for being so detailed about this here, is that we want to stress that rewriting is a three-phase process consisting of *matching* (decomposing a term into a context and a left-hand side), *replacement* (replacing the left-hand side by the corresponding right-hand side), and *substitution* (composing the context and the right-hand side into a term). Whereas usually emphasis is put on the second phase, here we will be interested in the third phase, substitution.

In the case of addition, the substitution phase is always simple since each rule of $\mathcal{A}$ is *linear*. That is, every variable occurs exactly once in both its sides (or not at all). For this reason, substitution can be thought of mathematically as a permutation. Implementing terms by graphs, and term rewrite rules by graph rewrite rules, so-called term graph rewriting [8]. permutation boils down to rewiring which can be performed in constant time. In Figure 1 the graph rewrite

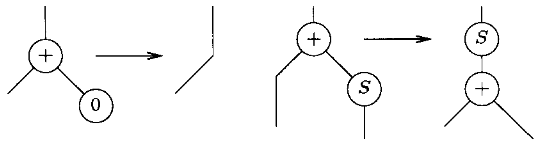


Figure 1. Graph rewrite rules for $\mathcal{A}$

system (GRS) corresponding to the TRS $\mathcal{A}$ is presented and Figure 2 displays the graph reduction corresponding to the reduction $\mathcal{R}$. Note that also a graph rewrite step can be decomposed into the three phases mentioned above. Typical other examples of linear term rewrite rules are the rules for commutativity and associativity:

$$\begin{aligned} x+y &\to y+x \\ (x+y)+z &\to x+(y+z) \end{aligned}$$

Unfortunately, not all term rewrite rules are linear.

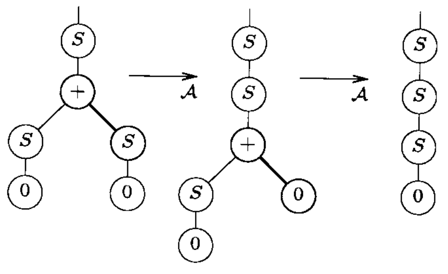


Figure 2. Graph reduction for $\mathcal{R}$

3. Non-linear substitution

Consider the extension $\mathcal{M}$ of the TRS $\mathcal{A}$ for addition, by the rules for multiplication:

$$\begin{aligned} x \times 0 &\to 0 \\ x \times S(y) &\to x + (x \times y) \end{aligned}$$

with typical reduction $\mathcal{S}$ given by

$$S(0) \times S(0) \to_{\mathcal{M}} S(0) + (S(0) \times 0) \to_{\mathcal{M}} S(0) + 0$$

Note that neither of the rules for multiplication is linear. The first is *erasing*: the variable x appears in its left-hand side, but not in its right-hand side. The second is *duplicating*: the variable x appears once in its left-hand side, but twice in its right-hand side. To represent replication we introduce the eraser node $\circledcirc$ and the duplicator node $\triangledown$ in the graph representation of these rules in Figure 3. The formal rewrite

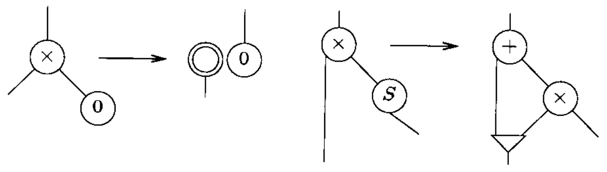


Figure 3. Graph rewrite rules for $\mathcal{M}$

rules for these replicator nodes will be presented in the next section, but the idea should be clear already from looking at the implementation of the reduction $\mathcal{S}$ in Figure 4: the argument connected to such a node is node-wise replicated the appropriate (0 or 2) number of times. The important point is that the substitution phase of reduction has become non-trivial; replication takes time linear in the size of the replicated argument. Indeed, substitution steps, indicated by the subscript x, form the majority of the steps in Figure 4. The idea is then to delay such steps.

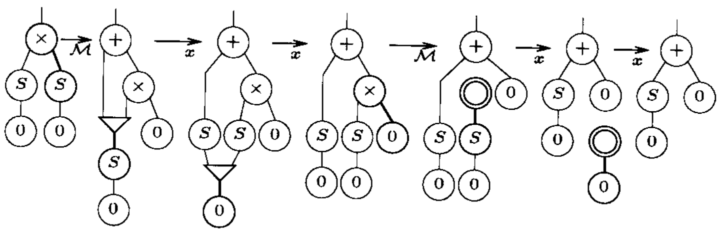


Figure 4. Graph reduction for $\mathcal{S}$

4. Explicit substitution rules

The graph rewrite rules for the eraser and duplicator are both instances of the two rule schemata in Figure 5. In a

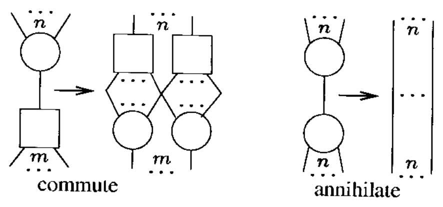


Figure 5. Schemata for explicit substitution

slogan: distinct symbols commute; identical symbols annihilate. All our rules for substitution operators will be instances of only these two simple schemata. In fact, for the moment just commutation suffices. In Figure 6 commutation is spelled out between on the one hand the replicators $\circledcirc$ and $\triangledown$ and on the other hand the function symbols 0 and S. Note that the right-hand side of the commutation rule for

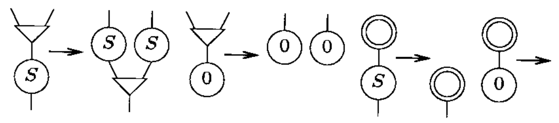


Figure 6. Commutation of $\triangledown$, $\circledcirc$ with $S, 0$

$\circledcirc$ and 0 in Figure 6 is the empty graph, as in this case both n and m in the commutation rule of Figure 5 are zero.

5. Delaying explicit substitution

In the naive graph rewriting implementation of $\mathcal{M}$ as in Figure 4, an $\mathcal{M}$-step is followed by a number of substitution steps until none is possible anymore, after which the next $\mathcal{M}$-step takes place, etc.. Having an explicit representation of substitution gives one the freedom to break this pattern. For instance, it is intuitively clear that duplicating a subterm for which it takes an expensive computation to yield a simple result is wasteful; computing the result first and then duplicating it saves half the time.

For terms, the delay of substitution is usually brought about by extending terms with the let-construct. For instance, applying a rule $x \times 2 \to x + x$ to $E \times 2$, yields let $x = E$ in $x + x$ instead of $E + E$, which is a good thing when E is expensive to compute. Hence, the let-construct can be viewed as an explicit substitution operator for terms.

For graphs, it suffices to break the pattern of reducing to substitution normal form after each rewrite step. Whereas up till now ordinary rewrite rules were only applied to graphs which were in fact trees, breaking the pattern leads to their application to graphs which are not trees.

Remark 1. In a *maximal* sharing discipline, as implemented in the ATerm library [3], *all* identical sub-terms are shared. That is, duplication is not just delayed but performed in the reverse direction to obtain maximal sharing.

6. Implementation

Stopping short of reaching the substitution normal form gives rise to the following adequacy questions, cf. [8]:

- Can one characterise the graphs representing terms?
- How do graph and term rewriting relate?

To give somewhat precise answers to these questions, it is useful to introduce a bit of notation. Let $\mathfrak{G}(t)$ denote the directed graph (in fact, tree) corresponding to the term t, obtained by directing all edges downward. Letting $\mathfrak{T}(G)$ denote the (unique if any) term t such that $\mathfrak{G}(t)$ is the substitution normal form of G, we have that $\mathfrak{T} \circ \mathfrak{G}$ is the identity on terms. The standard answer to the first question then is: representing graphs are directed and acyclic (dags). An equivalent characterisation in terms of substitution is:

graphs whose substitution normal form is a finite tree.

Uniqueness of substitution normal forms follows from confluence which holds since the substitution rules are orthogonal to one another; they constitute an interaction net [6]. A first answer to the second question is:

Lemma 2. ***(commutation)*** *If $G \to H$, then $\mathfrak{T}(G) \multimap\!\!\!\to \mathfrak{T}(H)$, where $\multimap\!\!\!\to$ denotes multi-step reduction, contracting a number of redexes in a term simultaneously.*

(progress) *If $\mathfrak{T}(G) \to s$, then there exist G', H such that $G \twoheadrightarrow_x G' \to H$, with the corresponding multi-step $\mathfrak{T}(G') \multimap\!\!\!\to \mathfrak{T}(H)$ contracting at least the redex contracted by $\mathfrak{T}(G) \to s$ (note that $\mathfrak{T}(G') = \mathfrak{T}(G)$).*

Remark 3. Depending on one's needs more stringent conditions can be put on the relationship, e.g. that graph rewriting of $\mathfrak{G}(t)$ should terminate if term rewriting of t does so.

7. Cyclic substitution

Dropping the finiteness condition in the characterisation of representing graphs above allows for the implementation of (potentially) infinite terms. To see this, consider ones, the infinite streams of 1s

$$1{:}1{:}1{:}1{:}\ldots$$

For terms, this infinite stream can be brought about by means of the letrec-construct

$$\mathsf{letrec}\; x = 1{:}x \;\mathsf{in}\; x$$

For graphs, it suffices to forget the finiteness condition on their substitution normal form in the characterisation above, as then the stream can be represented as the graph on the left in Figure 7. Indeed, computing the substitution

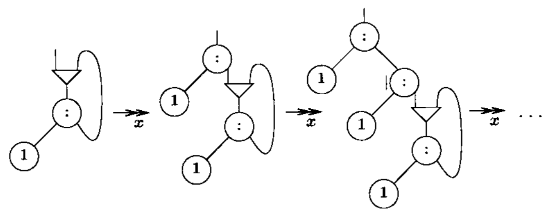


Figure 7. Infinite substitution normal form

normal form of this graph yields 'in the limit' an infinite tree representing the infinite stream of 1s as suggested in the figure. The results of the previous section for acyclic substitutions should extend to this cyclic case. We expect the implementation Lemma 2 can be shown by extending the theory of infinitary rewriting [9] from terms to graphs.

Note that we employ the same rule schemata for computing substitutions as before, and that combining them with an orthogonal TRS yields a combined system which is orthogonal, hence confluent. This is a bit surprising as it is well-known that collapsing on the one hand all the gs and on the other hand all the fs, in the infinite term $f(g(f(g(\ldots))))$ with respect to the orthogonal term rewrite rules $g(x) \to x$ and $f(x) \to x$, yields infinite terms $f(f(\ldots))$ and $g(g(\ldots))$ which are *not* joinable in the infinitary TRS. However, for their cyclic representation this is not a problem as shown in Figure 8; a so-called vicious circle [6] serves as the common reduct. A vicious circle is intuitively meaningless [9], but that's not needed here.

Remark 4. All infinite terms above represented by finite graphs are regular. Allowing the context-free substitutions to be introduced in the next section, non-regular ones such as the stream of natural numbers can be represented too.

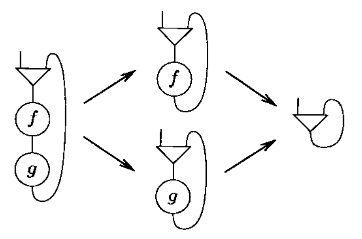


Figure 8. Confluence using a vicious circle

8 Scoping substitution

We now turn to implementing substitution for terms having binding symbols such as λ, $\forall$, $\sum$ etc. in an atomic manner. We treat the particular case of implementing β-reduction, i.e. substitution, for the λ-calculus [2]. As running example we take the Church-numeral $\underline{2}$ given by

$$\lambda x.\lambda y.x(xy)$$

First, we switch to the nameless λ-terms of [4], where each variable is replaced by a (unary) natural number indicating by which λ above it in the syntax tree the variable was bound (counting from zero). The representation of $\underline{2}$ is

$$\lambda\lambda(\mathsf{S}0)((\mathsf{S}0)0)$$

Second, we reinterpret a successor S as an explicit scope operator [5]; in a slogan: S stands for scope. The idea is illustrated in Figure 9, displaying from left to right, the syntax

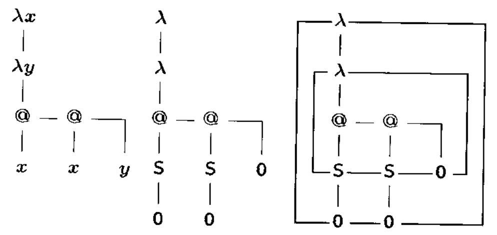


Figure 9. Reinterpreting successor as scope

tree of the ordinary λ-term $\underline{2}$, the syntax tree of its nameless version, and that tree again with scopes explicitly indicated by boxes. The boxes show that *binding* for an ordinary λ-term corresponds to *matching* for its nameless version: every S corresponds to a unique matching λ; any node below the S is *out of scope* of the λ, i.e. will not be affected by a substitution for its variable. Thus, λ-terms can be seen as *context-free* trees. The idea is to introduce the *scope* operator S into to our alphabet of substitution operators and implement substitution such that the matching structure is preserved. Unfortunately, this does not quite work because during β-reduction the neat nesting structure of boxes will be lost, they may partially overlap, and we must for each scope- and duplication-node individually keep track of how deep it is nested. To that end, we also index our operators as $\sqcup_i$ and $\triangledown_i$ for arbitrary depth i, setting $\triangledown = \triangledown_0$ and $\mathsf{S} = \sqcup_0$.

9. Translating λ-terms

We present an inductive translation of closed λ-terms into graphs built out of the explicit substitution operators and the function symbols λ (*abstraction*) and @ (*application*). Here a term t is *closed* if $0 \vdash t$ in the following inference system (to be read top–down)

$$\dfrac{\mathsf{S}i\vdash 0}{}\,0 \qquad \dfrac{\mathsf{S}i\vdash \mathsf{S}t}{i\vdash t}\,\mathsf{S} \qquad \dfrac{i\vdash \lambda t}{\mathsf{S}i\vdash t}\,\lambda \qquad \dfrac{i\vdash t_1t_2}{i\vdash t_1 \quad i\vdash t_2}\,@$$

The nameless term $\underline{2}$ is closed, as shown in Figure 10. A

$$\cfrac{0\vdash\lambda\lambda(\mathsf{S}0)((\mathsf{S}0)0)}{\cfrac{\mathsf{S}0\vdash\lambda(\mathsf{S}0)((\mathsf{S}0)0)}{\cfrac{\mathsf{SS}0\vdash(\mathsf{S}0)((\mathsf{S}0)0)}{\cfrac{\mathsf{SS}0\vdash\mathsf{S}0}{\cfrac{\mathsf{S}0\vdash 0}{}\,0}\,\mathsf{S} \quad \cfrac{\mathsf{SS}0\vdash(\mathsf{S}0)0}{\cfrac{\mathsf{SS}0\vdash\mathsf{S}0}{\cfrac{\mathsf{S}0\vdash 0}{}\,0}\,\mathsf{S} \quad \cfrac{\mathsf{S}0\vdash 0}{}\,0}\,@}\,@}\,\lambda}\,\lambda$$

Figure 10. Derivation showing $\underline{2}$ is closed

well-formed term $i \vdash t$ is mapped to a graph having $i + 1$ free ports, which is defined by induction and cases (0, S, λ, and @) on the derivation, in Figure 11. Here a number i next

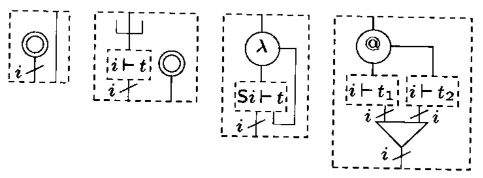


Figure 11. From λ-terms to graphs.

to a slashed edge represents that in fact the edge is a 'bus' consisting of i edges (connected to an appropriate number of nodes). Figure 12 shows the application to $\underline{2}$.

10 Implementing β-reduction

β-reduction is implemented by the rule in Figure 13. As before substitution is dealt with by the two rule schemata of Figure 5 (now annihilate is needed). In addition, indices need to be updated, where an *update* is an increment of the index i (if any) of a substitution operator, which takes place iff the other symbol is either λ or $\sqcup_j$, with $i \geq j$.

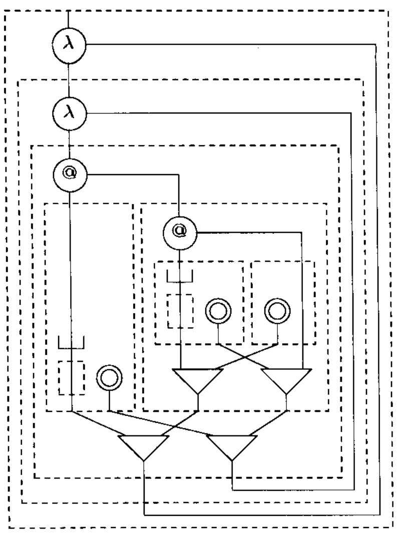


Figure 12. The graph translation of $\underline{2}$.

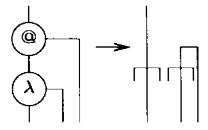


Figure 13. Translation of β-reduction rule

To get a flavour of this decomposition of β-reduction into atomic steps, in Figure 14 a part of the reduction of (an optimised translation of) $\underline{2}\,\underline{2}$ to graph normal form is shown. Although each of the individual steps is simple, it is easy to lose track of what is really happening, because there are so many steps. Recalling that application of Church-numerals is exponentiation, the final graph displayed should be a representation of the Church-numeral $\underline{4}$. Indeed it is. For an explanation as to why see [7]. There it is also shown that, as for the first-order case above, the implementation is adequate (Lemma 2). Again, the proof of adequacy does not depend on acyclicity, so should generalise to cyclic λ-terms.

11. Conclusion

We have presented an implementation of term rewriting based on keeping a clear distinction between on the one hand the implementation of substitution (the *substitution calculus* in the terminology of [10]), and on the other hand the implementation of operations on terms (here: the term rewrite rules). We have illustrated this for both the acyclic as well as the cyclic case, for first-order term rewriting and the λ-calculus. The atomic decomposition of substitution presented is simple (three operators), easy to im-

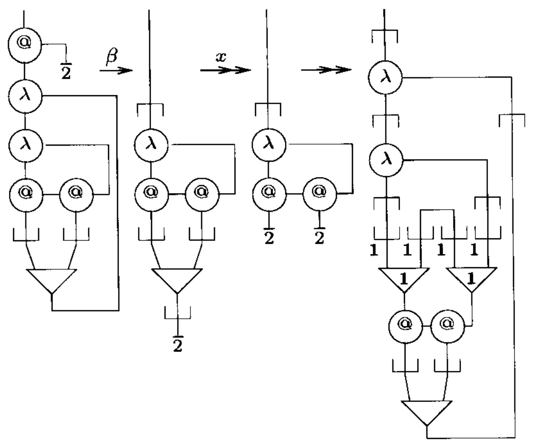


Figure 14. Reduction of $\underline{2}\,\underline{2}$

plement (two rule schemata), versatile (both acyclic and cylic), and intuitive (erasure, duplication and scoping are found in many contexts). We conclude with mentioning two possible applications. Representing proofs: the graphs can be seen as atomic decompositions of (the box in) Girard's proof nets for multiplicative exponential linear logic. Implementing functional programming: the implementation of β-reduction is in fact optimal in the sense of [1]. It would be interesting to combine this with other techniques.

Acknowledgements I have benefited from discussions over the years with co-authors, colleagues and students on the topic of this note, for which I thank them.

References

[1] A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages*. CUP, 1998.

[2] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, 1984.

[3] M. v. d. Brand, H. de Jong, P. Klint, and P. Olivier. Efficient annotated terms. *Software – Practice & Experience*, 30:259–291, 2000.

[4] N. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation. *Indagationes Mathematicae*, 34:381–392, 1972.

[5] D. Hendriks and V. v. Oostrom. ⅄. In *CADE 19*, volume 2741 of *LNAI*, pages 136–150. Springer, 2003.

[6] Y. Lafont. Interaction nets. In *POPL 17*, pages 95–108. ACM Press, 1990.

[7] V. v. Oostrom, K.-J. v. d. Looij, and M. Zwitserlood.]. Draft. Available from the first author's homepage, 2004.

[8] M. Sleep, M. Plasmeijer, and M. van Eekelen, editors. *Term Graph Rewriting*. John Wiley & Sons, 1993.

[9] Terese. *Term Rewriting Systems*. CUP, 2003.

[10] V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, Vrije Universiteit, Amsterdam, 1994.