Nieuwsbrief van de Nederlandse Vereniging voor Theoretische Informatica

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

Inhoudsopgave

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

1 Van de Redactie

Beste NVTI-leden,

Graag bieden wij u hierbij het vierde nummer aan van de jaarlijkse NVTI-Nieuwsbrief. Bij het samenstellen hebben we weer de formule van de vorige drie nummers gevolgd. Zo vindt u naast het programma van de jaarlijkse Theoriedag en de bijgewerkte ledenlijst ook weer enkele bijdragen van collega's met een korte inleiding in hun speciale gebied van expertise. Evenals voorgaande jaren zouden deze Nieuwsbrief en de Theoriedag niet tot stand hebben kunnen komen zonder de financiele steun van onze sponsors: EW NWO, Elsevier Publishing Company, en de onderzoekscholen IPA, SIKS en OZL. Namens de NVTI gemeenschap onze hartelijke dank voor deze middelen die ons voortbestaan mogelijk maken!

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

2 Samenstelling Bestuur

Prof.dr. J.C.M. Baeten (TUE)
Dr. H.L. Bodlaender (UU)
Prof.dr. J.W. Klop (VUA/CWI) voorzitter
Prof.dr. J.N. Kok (RUL) Prof.dr. J.-J.Ch. Meyer (UU)
Prof.dr. G.R. Renardel de Lavalette (RUG)
Prof.dr. G. Rozenberg (RUL)
Dr. J.J.M.M. Rutten (CWI) secretaris
Dr. J. Torenvliet (UvA)

3 Van de voorzitter

Geacht NVTI-lid,

In het afgelopen jaar is ons Bestuur met enkele nieuwe leden uitgebreid, om een betere overdekking van geografie en onderwerpen te krijgen. Ook dit jaar heeft het Bestuur zich beijverd om een interessante Theoriedag te organiseren. Deze zal gehouden worden op vrijdag 10 maart, op een voor ons nieuwe locatie (zie programma in dit nummer). We hopen en vertrouwen erop dat het programma voor velen van u interessant is. Graag tot ziens op 10 maart in Utrecht!

Jan Willem Klop, voorzitter NVTI

4 Theoriedag 2000

Vrijdag 10 maart 2000, Vergadercentrum La Vie, Utrecht

Het is ons een genoegen u uit te nodigen tot het bijwonen van de Theoriedag 2000 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 2000 zal gehouden worden op vrijdag 10 maart 2000, in:

Vergadercentrum La Vie, La Viestraat 351, Utrecht

(nabij CS Utrecht, op tien minuten loopafstand van de trein), en is een voortzetting van de reeks jaarlijkse bijeenkomsten van de NVTI die vijf jaar geleden met de oprichtingsbijeenkomst begon.

Evenals vorige jaren hebben wij een aantal prominente sprekers uit binnen- en buitenland bereid gevonden deze dag gestalte te geven 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, alsmede presentaties van de onderzoekscholen.

Programma

09.30-10.00: Ontvangst met koffie

10.00-10.10: Opening

10.10-11.00: Lezing Prof.dr. A. Pnueli (The John von Neumann Minerva Center for Verification of Reactive Systems, Weizmann Institute of Science, Israel) Titel: Uniform Verification of Parameterized Networks

11.00-11.30: Koffie

11.30-12.20: Lezing Prof.dr. E. Brinksma (Universiteit Twente) Titel: Stochastic Process Algebras: Linking Process Descriptions with Performance

12.20-12.50: Presentatie Onderzoeksscholen (OZL, IPA, SIKS)

12.50-14.10: Lunch (Zie beneden voor registratie)

14.10-15.00: Lezing Prof.dr. S. Micali (MIT, Boston) Titel: The Quest for Efficient Proofs

15.00-15.20: Thee

15.20-16.10: Lezing Dr.ir. H. Te Riele (CWI, Amsterdam) Titel: Factoring algorithms and their complexity, with application to cryptography

16.10-16.40: Algemene ledenvergadering NVTI

Abstracts van de voordrachten

Uniform Verification of Parameterized Networks

Prof.dr. A. Pnueli
(The John von Neumann Minerva Center for Verification of Reactive Systems, Weizmann Institute of Science, Israel)

A parameterized network is a system consisting of many interconnected processes, whose size is a parameter. A typical example is a set of processes situated on a ring whose size is a parameter N. Usually, each individual process in the network is a finite-set program communicating with its close neighbors.

We can always verify properties of particular instances of such a network, e.g. consider a process-ring of sizes 5, 7, and 11, by automatic model-checking techniques. However this is not in general sufficient in order to conclude that the properties are valid over ALL instances of the system. Uniform verification of such systems attempts to establish in one verification effort the validity of the considered properties for EVERY value of the parameter N.

In the talk, we will survey several approaches to the solution of this problem. The three approaches to be reviewed are the formation of Network Invariants, Finitary Abstraction, and Regular Symbolic Model-Checking, which performs symbolic model checking over a rich assertional language which can capture infinite sets of states.

Stochastic Process Algebras: Linking Process Descriptions with Performance
Prof.dr. E. Brinksma
(Universiteit Twente)

In this presentation we will give an overview of the research on stochastic process algebras, a branch of process algebra that has developed over the last decade. Like ordinary process algebra, stochastic process algebra (SPA) provides a compositional model for the description and analysis of complex distributed systems, such as network protocol systems. At the same time SPAs are extended with stochastic features to enable the compositional and systematic derivation of performance models from such descriptions. We will discuss the motivation and potential benefits of the use of SPAs, and present the main conceptual issues in their development. In particular, we will present so-called Markovian process algebras that allow for performance analysis in terms of Continuous Time Markov Chains (CTMC), as well as a non-Markovian process algebra enabling the use of more general performance models.

The Quest for Efficient Proofs
Prof.dr. S. Micali
(MIT, Boston)

Efficiently proving the verity of a statement is of crucial importance, but what should EFFICIENTLY PROVING mean? In the last three decades, much effort has been devoted to answer this question, yielding the notions of NP, Interactive Proofs, and Probabilistically Checkable Proofs.

Computationally Sound (CS) Proofs provide a new answer to this everlasting question. Informally, a CS proof of a statement S consists of a short string which (1) is as easy to find as possible, (2) is very easy to verify, and (3) computationally guarantees the verity of S: In essence, CS proofs of false statements either do not exist or are practically impossible to find.

CS proofs provide a quite effective way to handle membership in computationally hard languages (such as Co-NP complete ones) and correctness of long computations.

Factoring algorithms and their complexity, with application to cryptography,
Dr.ir. H. Te Riele
(CWI, Amsterdam)

In Volume 2: "Seminumerical algorithms" of Knuth's "The art of computer programming" (first edition appeared in 1969) a prominent place has been reserved for the problem of factoring large numbers. The publication, in 1978, of the RSA public-key cryptographic system has stimulated renewed interest in factoring, because the security of RSA depends on the difficulty of factoring large numbers and because the computational complexity of factoring large numbers is still unknown.

As a result, the world record for factoring (difficult) large numbers has been pushed forward from about 40 digits in 1969 to 155 digits in 1999. We will explain the algorithmic and computer developments behind this progress. In addition, details will be given of the algorithmic and computational effort invested in the current factoring world record: a 512-bits' (= 155 decimal digits) RSA key. Such keys are widely used to protect E-commerce on the Internet, and in SSL handshake protocols.

Lidmaatschap NVTI

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

Lunchdeelname

Het is mogelijk aan een georganiseerde lunch deel te nemen; hiervoor is aanmelding verplicht. Dit kan per email of telefonisch bij Mieke Bruné (mieke@cwi.nl, 020-592 4249), tot een week tevoren (3 maart). De kosten kunnen ter plaatse voldaan worden; deze bedragen (ongeveer) f 21,50. 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.

5 Mededelingen van de onderzoekscholen

Hieronder volgen korte beschrijvingen van de onderzoekscholen:

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

5.1 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 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. In 1999, the composition of IPA was unchanged. Researchers from eight universities (University of Nijmegen, Leiden University, Eindhoven University of Technology, University of Twente, Utrecht University, University of Groningen, Vrije Universiteit Amsterdam, and the University of Amsterdam) , the CWI and Philips Research (Eindhoven) participated.

The curriculum of IPA has two parts; a series of three 'basic courses' designed to provide Ph.D. students with an overview of research in each of the three main areas. In 1999 the basic course on Formal Methods was given. Besides these courses with fixed subjects there are two multi-day events per year, the Spring-days and Fall-days, which focus on upcoming new subjects. In 1999 the Spring-days were dedicated to Probabilistic Methods in Computer Science, and the Fall-days to Component-based Software Development.

Besides that, IPA was involved in the organisation of international activities. Working together with the research schools BRICS (Denmark) and TUCS (Finland) in the European Educational Forum (EEF), the Synergos Summerschools on 'Logic and Computation' (Herriot-Watt University, Edinburgh) and 'Semantics of Computation' (University of Aarhus, Denmark) were realized.

Ph.D. Defenses in 1999

E. Voermans
Inductive Datatypes with Laws and Subtyping – A Relational Model
Faculty of Mathematics and Computing Science, TUE
H. ter Doest
Towards Probabilistic Unification-based Parsing
Faculty of Computer Science, UT
J.P.L. Segers
Algorithms for the Simulation of Surface Processes
Faculty of Mathematics and Computing Science, TUE
E.I. Barakova
Learning Reliability: a Study on Indecisiveness in Sample Selection
Faculty of Mathematics and Natural Sciences, RUG
M.P. Bodlaender
Scheduler Optimization in Real-Time Distributed Databases
Faculty of Mathematics and Computing Science, TUE

M.A. Reniers
Message Sequence Chart: Syntax and Semantics
Faculty of Mathematics and Computing Science, TUE
J.P. Warners
Nonlinear approaches to satisfiability problems
Faculty of Mathematics and Computing Science, TUE
J.M.T. Romijn
Analysing Industrial Protocols with Formal Methods
Faculty of Computer Science, UT
P.R. D'Argenio
Algebras and Automata for Timed and Stochastic Systems
Faculty of Computer Science, UT
G. Fábián
A Language and Simulator for Hybrid Systems
Faculty of Mechanical Engineering, TUE
J. Zwanenburg
Object-Oriented Concepts and Proof Rules
Faculty of Mathematics and Computer Science, TUE
R.S. Venema
Aspects of an Integrated Neural Prediction System
Faculty of Mathematics and Natural Sciences, RUG
J. Saraiva
A Purely Functional Implementation of Attribute Grammars
Faculty of Mathematics and Computer Science, UU
R. Schiefer
Viper, A Visualisation Tool for Parallel Progam Construction

Faculty of Mathematics and Computer Science, TUE

Activities in 2000

During the past four years, IPA has organised a series of "Synergos Summer Schools" working together with BRICS, TUCS and UKII in the EEF. In the next four years this activity will be continued: from 2000 - 2004 the EEF will organise two series of four Summer Schools, sponsored by the European Community in the IHP-programme of the fifth framework. The EEF Foundations Series will provide in-depth foundational knowledge to young researchers on topics which are traditionally at the core of research in EEF (Deduction and Theorem Proving, Logical Methods, Specification Refinement and Verification, Concurrency). The EEF Trends Series will focus on important topics that have recently emerged (Probabilistic Methods, Software Architecture, Massive Data Sets, Mobile Computing), and tries to distill from them "what is really going on". The first event in the Trends Series will be staged by IPA this summer.

On the homefront, IPA will organise the Spring-days (the topic: UML), the Fall-days and two standard courses: Software Technology and Algorithmics & Complexity, each covering one of IPA's main research areas. To stay informed on the activities of IPA, you can download the IPA newsletter (in Dutch) from our Web-site.

IPA Spring Days on UML
April 26 - 28, 2000, Motel Eindhoven, Eindhoven, The Netherlands.

UML (Unified Modeling Language) has rapidly become a de facto standard in the software industry. The existence of a widely used and powerful language offers new possibilities for connecting academic research to industrial practice. This seminar seeks to explore these possibilities. Further information will become available through our website.
see: http://www.win.tue.nl/cs/ipa/activities/lentedagen2000.html

EEF TrendSchool on Formal Methods and Performance Analysis
July 3 - 7, 2000, Golden Tulip Valmonte, Nijmegen, The Netherlands.
Working together with BRICS, TUCS and UKII in the EEF, IPA organises the first in a series of four EEF schools on Trends in computer science, sponsored by the European Community in the IHP programme of the fifth framework. This event is dedicated to Probabilistic Methods.
see: http://fmt.cs.utwente.nl/conferences/fmpa2k/

Addresses

Visiting address
Eindhoven University of Technology
Main Building HG 7.17
Den Dolech 2
5612 AZ Eindhoven
The Netherlands

Postal address
IPA, Fac. of Math. and Comp. Sci.
Eindhoven University of Technology
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 http://www.win.tue.nl/cs/ipa/

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

The Dutch Research School in Logic (OzsL) is active in three main areas: mathematical logic, logic in linguistics and philosophy, and logic in computer science. Formal participants in the School are the University of Amsterdam, the Free University in Amsterdam, the University of Utrecht, the University of Groningen, and Tilburg University. In addition, there are numerous associate members, to cater for the need of those who have active scientific links with the OzsL community, while political reasons argue against full participation. The general policy of the school is to foster cooperation rather than competition with neighbouring schools, and associate membership is open for all our neighbours.

Recently, prof Göran Sundholm (philosophical logic, University of Leiden) and prof Rob van der Sandt (philosophical logic, Nijmegen University) joined OzsL.

International cooperation agreements exist with Stanford University, the University of Edinburgh, the University of the Saarland, and the University of Stuttgart. Funding is available for visitor exchanges within this international network, and regular international workshops take place within the network.

The Ph.D. courses offered by the school fall in two categories: courses that are part of the 'school week curriculum', and master classes. School weeks are offered twice a year, in Spring and in Autumn. To give an idea of the contents, here is a recent sample:

Autumn 1999 School Week, Amsterdam The curriculum consisted of four tutorials, in logic and model theory, logic and information flow analysis, logic and agent technology, and history of logic. In more detail:

- Ehrenfeucht's Game, by Kees Doets,
- Information Flow From Source to Sink, by Jeroen Groenendijk, Reinhard Muskens, Robert van Rooy and Albert Visser
- Intelligent Agents, by Catholijn Jonker, John-Jules Meyer, Jan Treur, Rineke Verbrugge,
- The Development of Logic from Bolzano to Heyting and Gentzen, by Göran Sundholm.

Further details can be found in the web archive of the School, at address
`http://www.ozsl.uva.nl/archive.html`.

In combination with the Autumn School Week the yearly Accolade Event takes place: an occasion where PhD students within the school present their work to the outside world in an informal setting. Somewhat more than a year ago it was decided to change the character of this event. It was felt by the OzsL Board that previous Accolade meetings had to strike an uneasy compromise between informing the community about how individual Ph.D. projects were progressing on one hand, and functioning as an internal workshop where our Ph.D. students could fine-tune their conference presentation skills in a friendly and supportive setting. Therefore the Board decided to separate these two functions, and to relegate the second to the regular colloquia.

Accolade New Style now has as its single aim to inform the Dutch logic community about how the Ph.D. projects within the School are going, irrespective of whether these projects are in an initial, intermediate or final stage of research. Accolade New Style was held for the second time last December. The change in style is generally felt to be an improvement, and participants are enthousiastic about the formula, where they get useful response in a supportive setting.

Another recent innovation is the organisation of a complement event to Accolade in Spring. This 'Accolade for Adults' (also known under various silly nicknames) is meant to create an opportunity for staff members within the School to give brief outlines of their research, for an audience consisting of their colleagues and the PhD students of the School.

OzsL issues LIN ('Logic in the Netherlands'), a newsletter for the Dutch logic community in the broadest possible sense, that appears at rather irregular intervals, both on paper and electronically. Subscription is free of charge; please send an email to the OzsL Office Manager dr Peter Blok at `pblok@wins.uva.nl` to subsribe. Further information about the school and its activities is available electronically, at `http://www.ozsl.uva.nl`.

5.3 School voor Informatie- en KennisSystemen SIKS, door: R.J.C.M. Starmans

Inleiding

De School for Information and Knowledge Systems (SIKS) zag begin 1996 het licht. Het initiatief tot de oprichting was drie jaar eerder genomen door een groep onderzoekers op het terrein van kunstmatige intelligentie, database theorie en software engineering. Zij trokken zich enige dagen terug op het Vlaamse platteland om de haalbaarheid te onderzoeken van een nieuwe onderzoekschool. Deze zou zich moeten toeleggen op fundamenteel en toepassingsgericht onderzoek op het gebied van de informatica, meer in het bijzonder op het terrein van informatie- en kennissystemen. Daarnaast moest zij borg staan voor een hoogwaardige promovendi-opleiding en bovenal een eigen en herkenbare plaats innemen ten opzichte van andere samenwerkings-verbanden binnen de informatica in Nederland.
Het benodigde universitaire draagvlak bleek aanwezig en de initiatieven resulteerden in 1994 in een formeel voorstel tot oprichting van onderzoekschool SIKS. Twee jaar later werd conform de WHW en overeenkomstig het verkavelingsplan van de Stichting Informatica Onderzoek Nederland (SION) een interuniversitaire samenwerkingsovereenkomst gesloten. SIKS kon van start onder het penvoerderschap van de Vrije Universiteit van Amsterdam. De Universiteit Utrecht, de Technische Universiteit Delft, de Technische Universiteit Eindhoven, de Universiteit Twente, de Rijksuniversiteit Leiden en de Universiteit Maastricht traden eveneens toe, korte tijd later gevolgd door de Universiteit van Amsterdam. Ook met het Centrum voor Wiskunde en Informatica, de Erasmus Universiteit Rotterdam en de Katholieke Universiteit Brabant werden samenwerkingsverbanden aangegaan.

KNAW-erkenning in 1998

Al spoedig groeide het besef dat accreditatie door de Erkenningscommissie van de KNAW een belangrijke waarborg vormt voor de continuïteit van de school. Een grondige herbezinning op

de positie van de school was hiertoe nodig. Zo werd bij voorbeeld de missie van SIKS, zoals indertijd verwoord in het oorspronkelijke oprichtingsvoorstel, ingrijpend herzien. Ook koos SIKS er nadrukkelijk voor het onderzoek te centreren rond een vijftal research foci: knowledge science, coöperatieve systemen, requirements engineering en formele specificatie van IKS, multimedia en tot slot architecturen van IKS. De keuze en invulling van deze aandachtsgebieden werden ingegeven door interne ontwikkelingen binnen de wetenschapsgebieden waarop SIKS actief is, maar sloten bovendien nauw aan bij externe beleidsinitiatieven en onderzoeksagenda's van met name NWO en SION. Zoals bekend slaagde SIKS in haar opzet. In mei 1998 was de erkenning door de KNAW een feit.

Activiteiten in 1999

SIKS begon 1999 met de aanstelling van een nieuwe coördinator. Richard Starmans volgde Koen Versmissen op per 1 januari 1999. Ook werd hard gewerkt om het onderwijsprogramma verder van de grond te krijgen, het onderzoeksprogramma te versterken, de interne organisatie van SIKS te verbeteren en bovenal de herkenbaarheid van de school te bevorderen door een duidelijke positionering ten opzichte van de twee andere Nederlandse onderzoekscholen op het terrein van de informatica. Een aantal activiteiten uit 1999 zal hier kort de revue passeren.

Vier promovendi-cursussen

Om te beginnen werden vier landelijke promovendicursussen georganiseerd, die ook voor promovendi van andere scholen en (in beperkte mate) voor externe deelnemers werden opengesteld. Van 31 mei tot en met 4 juni 1999 vonden in conferentiecentrum Woudschoten te Zeist de promovendicursussen Combinatorische Methoden en Intelligent Systems plaats. Van 29 november tot en met 3 december volgden in Hotel Apeldoorn Systeemmodelleren en Kennismodelleren. Deze vier cursussen maken allen deel uit van het verplichte basisprogramma van de school. De cursussen werden in totaal door ruim 50 promovendi gevolgd.

Promoties

De laatste jaren kon een groeiend aantal promoties binnen de school worden gesignaleerd, zowel van eerste-geldstroom promovendi, als van promovendi die door de industrie worden bekostigd. Om die reden is SIKS in 1998 van start gegaan met een eigen dissertatie-reeks. In 1999 konden hierin de volgende SIKS-proefschriften worden bijgeschreven.

99-1 Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products
Promotor: prof.dr. J. Treur
Co-promotor: dr.ir. M. Willems
Promotie: 11 mei 1999

99-2 Rob Potharst (EUR)
Classification using decision trees and neural nets
Promotor: prof. Dr. A. de Bruin
Co-promotor: dr. J.C. Bioch
Promotie: 4 juni 1999

99-3 Don Beal (Queen Mary and Westfield College)
The Nature of Minimax Search
Promotor: Prof.dr. H.J.van den Herik
Promotie: 11 juni 1999

99-4 Jacques Penders (KPN Research)
The practical Art of Moving Physical Objects
Promotor: Prof.dr. H.J. van den Herik
Co-promotor: dr. P.J. Braspenning
Promotie: 11 juni 1999

99-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems
Promotor: Prof.Dr. R.A. Meersman
Co-promotor: Dr. H. Weigand
Promotie: 1 oktober 1999

99-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems
Promotor: prof.dr. J. Treur
Copromotor: dr. F.M.T. Brazier
Promotie: 30 september 1999

99-7 David Spelt (UT)
Verification support for object database design
Promotor: Prof. Dr. P.M.G. Apers
Assistent promotor: Dr. H. Balsters
Promotie: 10 september 1999

99-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation
Promotor: Prof. dr. H.J. van den Herik
Co-promotor: Dr. P.J. Braspenning
Promotie: 3 december 1999

SIKS-dag 1999

Op 1 oktober vond de jaarlijkse SIKS dag plaats, de ontmoetingsdag van de bij SIKS aangesloten onderzoekers. Plaats van handeling was dit jaar Grand Hotel Krasnapolsky te Amsterdam. De bijeenkomst had niet alleen tot doel de KNAW-erkenning enige luister bij te zettten, maar gaf het bestuur tevens de gelegenheid vooruit te blikken op een aantal activiteiten die in de nabije toekomst worden georganiseerd. Ook vond er tijdens de dag een mini-symposium plaats, waarop Robert Meersman (VU Brussel), Roel Wieringa (UT) en Michael Wooldridge (Queen Mary and Westfield College, London) als gastspreker optraden.

Masterclasses en doctoraalconsortium

Voorts organiseerde SIKS op 1 en 2 November in Maastricht, voorafgaand aan BNAIC 99, een tweetal masterclasses voor promovendi en stafmedewerkers. Op maandag 1 november beet Jonathan Schaeffer (University of Alberta) de spits af met "Heuristic Search: Tools of the Trade". De tweede masterclass werd de volgende dag verzorgd door Tom Mitchell (Carnegie Mellon University) die de aanwezigen "An overview of Machine Learning" bood. Op dinsdag 2 november vond tot slot ook een doctoraalconsortium plaats. Vier promovendi gaven uiteenzettingen over hun onderzoeksresultaten dan wel onderzoeksplannen ten overstaan van een forum van stafleden. Elke SIKS-promovendus dient tijdens zijn opleiding ten minste 1 maal aan een doctoraalconsortium deel te nemen.

Samenwerking SIKS-bedrijfsleven

Ook is in 1999 een aanvang gemaakt met de in 1998 door de Erkenningscommissie aanbevolen samenwerking met het bedrijfsleven. Deze inspanningen moeten op zeer korte termijn leiden tot de inrichting van het Platform SIKS industrie (PSI). Tijdens de studiedagen van SIKS op 13 en 14 januari 2000 zijn hiertoe de eerste stappen gezet. Bij de bijeenkomst waren verschillende geïnteresseerde bedrijven aanwezig.

Plannen voor 2000

Oprichting Platform SIKS- Industrie De verwachting is dat de oprichting van het Platform in het voorjaar van 2000 gerealiseerd zal worden.

Electronisch Magazine SIKSTANT

SIKS hoopt op korte termijn met een eigen electronisch magazine SIKSTANT de deelnemers in de school en de externe relaties beter te informeren over projecten en andere ontwikkelingen binnen SIKS en haar naaste omgeving.

Onderwijsprogramma

Ook in 2000 zal een viertal promovendi-cursussen worden georganiseerd. Van 5 tot en met 9 juni staan Databases en Interactieve Media op het programma. In december zullen twee cursussen worden verzorgd waarvan de inhoud op dit moment nog niet bekend is. Master-classes zullen naar verwachting plaatsvinden eind augustus in Amsterdam en tijdens de BNAIC 2000 te Tilburg.

Research foci

De erkenningscommissie verwacht dat het onderzoek zich voor een belangrijk deel zal richten op thema's die verband houden met voornoemde research foci en meent dat dit de coherentie van het onderzoeksprogramma ten goede zal komen. Waar mogelijk zal daarom worden getracht wetenschappelijke en meer gespecialiseerde onderwijsactiviteiten zoveel mogelijk te laten aansluiten bij de research foci. Zo zal SIKS in het voorjaar van start gaan met een colloquium over coöperatieve systemen.

Voor meer informatie over SIKS en haar activiteiten kunt u de site van de onderzoekschool raadplegen (www.siks.nl) of contact opnemen met het Bureau van SIKS: 030-2534083, email: office@siks.nl.

6 Wetenschappelijke bijdragen

6.1 Perspectives on Integer Programming: Karen Aardal

Informatica Instituut, Universiteit Utrecht, Postbus 80089, 3508 TB Utrecht.
e-mail: aardal@cs.uu.nl. URL: http://www.cs.uu.nl/people/aardal/

Perspectives on Integer Programming

Karen Aardal*

In integer programming we want to minimize or maximize a linear objective function subject to a set of constraints formulated as linear equalities or inequalities. The decision variables are allowed to take integer values only. Mathematically we can formulate the integer programming problem as follows:

$$\min\{\boldsymbol{c}^T\boldsymbol{x} : \ \boldsymbol{x} \in P \cap \mathbb{Z}^n\} \tag{1}$$

where $P = \{\boldsymbol{x} \in \mathbb{R}^n : \ \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$. One typically assumes that the elements of the matrix $\boldsymbol{A}$ and the vectors $\boldsymbol{c}$ and $\boldsymbol{b}$ are rational. What makes this simple problem so interesting to consider? There are numerous theoretical questions one comes across when analyzing the problem, there are several algorithmic challenges surrounding the resolution, and many problems from practice can be formulated as integer programming problems. So, there is something for almost everybody! Integer programming is a true melting pot for mathematicians, computer scientists, engineers, production planners, economists, and many, many more. In this note I will try to sketch a few issues of theory, algorithms, and practice, and I also want to give my personal view on some interesting questions to consider in the future. Before starting I will describe an application of integer programming.

Example 1 In telecommunication there are many problems that can be formulated as integer programming problems. Here I will describe one that is known as the survivable network design problem (see e.g. Stoer, 1992). We are given a set of telephone offices, and the problem is to decide on how to connect the offices such that the obtained network is "survivable", and such that the total cost of placing the connections between the offices is minimized. In network terminology we refer to the telephone offices as the *nodes* of the network, and the connections between pairs of offices as the *edges* of the network. A survivable network means that for each pair of distinct nodes there should exist a given minimum number of edge-disjoint paths. The offices are partitioned into different categories. Typical categories are: main offices, which are given label 2; ordinary offices, which are given label 1; and optional offices, which are given label 0. We denote the label belonging to node s by r_s. The minimum number of edge-disjoint paths between a pair (s,t) of offices, denoted $q(s,t)$, is then the minimum of the two node labels, i.e., $q(s,t) := \min\{r_s, r_t\}$. To formulate this problem as an integer programming problem we introduce the decision variables x_e for each possible edge e in the network. Variable x_e takes value 1 if edge e is included in the network, and value 0 otherwise.

*Informatica Instituut, Universiteit Utrecht, Postbus 80089, 3508 TB Utrecht.
e-mail: aardal@cs.uu.nl. URL: http://www.cs.uu.nl/people/aardal/

The objective is to minimize:

$$\sum_{\text{all edges } e} c_e x_e$$

where c_e is the cost of including edge e. The constraints that ensure survivability can be modeled as:

$$\sum_{\{e:\ |e \cap W|=1\}} x_e \geq \text{con}(W) \quad \text{for all subsets } W \text{of the node set} V \text{ such that } \emptyset \neq W \neq V.$$

Here $\text{con}(W)$ is defined to be $\text{con}(W) := \max\{q(s,t) :\ s \in W,\ t \in V \setminus W\}$. We further need to impose integrality and bounds on the variables:

$$0 \leq x_e \leq 1 \quad \text{for all edges } e, \quad x_e \text{ integral for all edges } e.$$

■

For the reader interested in more literature, I recommend the textbooks by Schrijver (1986), Nemhauser and Wolsey (1988), and Grötschel, Lovász, and Schrijver (1988), and the recent survey articles by Marchand, Martin, Weismantel, and Wolsey (1999), and Aardal, Weismantel and Wolsey (1999).

1 How it all started

The field of integer programming was started by Ralph E. Gomory. After graduating at Princeton University and serving in the U.S. Navy he worked for the U.S. Navy as a consultant while holding a position at Princeton. The following is a quotation from an interview with Gomory (Aardal, 1996):

> "So, every few weeks I would go down there and consult with them on problems. One of these problems was actually an integer programming problem, but it was formulated as a linear programming problem, and the variables came out noninteger. This was very awkward since they represented the number of aircraft carriers. The answer 2.2 would not actually be so bad, but 0.6 is awful because you then have to ask if you need any aircraft carriers at all. As soon as I saw that I thought that in solving integer linear *equations* there is a diophantine way, so probably, in solving integer linear *inequalities* there should be a diophantine way as well. Of course I was wrong, but that is how I started my work. We needed the answer."

At that time Dantzig's (Dantzig, 1951) *simplex algorithm* for solving the *linear* programming problem

$$\min\{\boldsymbol{c}^T \boldsymbol{x} :\ \boldsymbol{x} \in P\} \tag{2}$$

was known and had caused a lot of excitement among mathematicians and economists, so it was natural to think about an algorithm, based on the simplex algorithm, for solving the integer programming problem (1). The simplex algorithm works as follows.

The set P is the set of vectors $\boldsymbol{x} \in \mathbb{R}^n$ satisfying a *finite* set of linear inequalities. Such a set is called a *polyhedron*. Notice that the linear programming problem (2) is a *relaxation* of the integer programming problem (1), which implies that $\min\{\boldsymbol{c}^T\boldsymbol{x} : \boldsymbol{x} \in P\} \leq \min\{\boldsymbol{c}^T\boldsymbol{x} : \boldsymbol{x} \in P \cap \mathbb{Z}^n\}$. We call this relaxation the *linear programming relaxation*, or LP-relaxation for short. It is easy to see that if we maximize or minimize a linear function over a polyhedron, then an optimal solution is found in an extreme point of the polyhedron if the objective value is bounded. Assume that we are considering a minimization problem. What the simplex algorithm does is to iterate from one extreme point of P to another such that the objective value does not increase. To find a initial extreme point is easy. If we consider the integer programming problem then the obvious problem in this context is that an extreme point of the polyhedron P is not necessarily integral. The idea of Gomory was to iteratively add new linear inequalities in order to get rid of "unnecessary" parts of P such that eventually the optimal solution to the resulting LP-relaxation becomes integral. The formulation of each of these linear inequalities was obtained from the optimal solution of the current LP-relaxation. Since each linear inequality is defined by a hyperplane, and since each of Gomory's inequalities is "cutting off" parts of P, his algorithm is called *Gomory's cutting plane algorithm* (Gomory himself called it "the fractional cut method"). Gomory (Gomory, 1958, 1960) proved that under some technical assumptions his algorithm terminates within a finite number of iterations with an optimal integer solution, or a proof that P does not contain an integer vector. Gomory later developed cutting planes for mixed integer problems, i.e., problems where a given subset of the variables may only take integer values, and where the remaining variables may take real values. Up to this day the idea of adding cutting planes to the inequalities defining P is the dominant technique used to solve integer and mixed integer programming problems.

Until quite recently Gomory's cutting plane algorithm had the reputation of not being very practical. Such a statement is made in many textbooks over the years. There are several reasons for this. First, the computers at that time were not very powerful, so solving even modest size linear programming problems, on which Gomory's algorithm relies, was not very easy. Second, the linear programming software did not let the user iteratively add new inequalities in an easy way, making cutting plane algorithms cumbersome to use. Third, the early implementations did exactly what Gomory said in his proof that the algorithm terminates in a finite number of steps: add one inequality at the time and reoptimize. First of all "finite" can be very large, and one would not like to add an unnecessarily large number of inequalities as the running time of the simplex algorithm in practice is depending more on the number of constraints in the formulation than the number of variables. Moreover, Gomory inequalities are typically dense, i.e., they contain many nonzero elements, which may result in numerical problems. Today, the computers are amazingly more powerful than fifty years ago, but this is true also for the linear programming solvers. Solving linear programming problems with a million of variables is possible nowadays. It is important to note that only about half of this "computational" progress made during the past couple of decades is due to faster computers. The rest is due to better algorithms and data structures, and to more efficient implementations. The improved algorithms and implementations can deal with numerical issues caused by the dense Gomory inequalities so the only issue left is then: how should we add the inequalities? One at the time? What if the convergence is

very slow? Recent work (see e.g. Balas et al., 1996) has shown that one should add more inequalities at each iteration, and that one should stop once the improvements of adding cuts is getting small. Instead of adding many weak inequalities it is better to partition the set of feasible solutions into smaller parts. The way to do this is by *branch-and-bound*, which we will describe very briefly below. In recent computational work Gomory inequalities, in particular the mixed-integer inequalities, have been used with great success, see further Marchand et al.

2 Modern cutting plane techniques

One of the questions that were, at least partially, inspired by the relative ineffectiveness of the early trials with Gomory's cutting planes is whether the Gomory inequalities are "strong" in the sense that they cut deep into the polyhedron P, without, of course, cutting off any integer feasible vectors. One way of looking at the problem is as follows: "I know how to solve linear programming problems, so why don't I try to find the ideal linear relaxation of my problem?" The best possible linear description of the feasible set $X = \{P \cap \mathbb{Z}^n\}$ is the convex hull of X, conv(X), see Figure 1. The set conv(X)

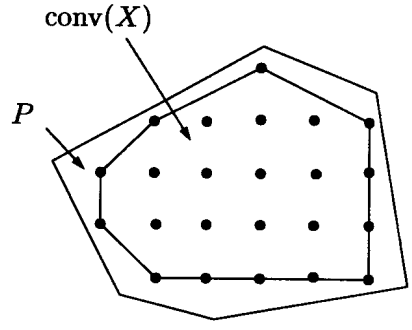

Figure 1: THE CONVEX HULL OF X.

is again a polyhedron, and moreover, each extreme point of conv(X) is integral! So, problem (1) is equivalent to:

$$\min\{\boldsymbol{c}^T\boldsymbol{x} : \ \boldsymbol{x} \in \text{conv}(X)\} \tag{3}$$

Note that problem (3) is a linear programming problem, which is polynomially solvable. Readers familiar with complexity theory will think that I am either foolish, or have found a way of proving a large result by this statement. But, there is of course a catch here. Finding an explicit description of conv(X) is hard. This will be discussed in the next section.

To solve a specific instance we of course do not need to know the complete description of conv(X). A good description of conv(X) in the neighborhood of the optimal solution is often good enough. This inspired several researchers in the seventies and onwards to consider classes of inequalities for a certain type of feasible set X and to try to prove that they are necessary in the description of conv(X). Such inequalities are called *facet defining*. One of the early computational breakthroughs was made by Grötschel

(Grötschel, 1980) who solved a 120-city traveling salesman problem (he actually solved the instance in 1977). The traveling salesman problem is the problem of finding a minimum length Hamiltonian cycle, i.e. a cycle visiting each node exactly once, in an undirected graph. Grötschel added linear inequalities that belonged to facet-defining classes of inequalities for the traveling salesman polyhedron. The largest traveling salesman instance solved at that point was the famous 49-city instance that was solved by Dantzig, Fulkerson, and Johnson (Dantzig et al., 1954). Dantzig and his colleagues used cutting planes that they developed by analyzing the specific fractional solutions that they obtained after iteratively adding cutting planes. The cutting planes were neither Gomory cuts, nor were they proved to be facet defining. Nevertheless, their work is seminal in the development of integer programming. It is important to keep in mind that a 49-city traveling salesman instance was huge at that time as it contains more than 1,100 decision variables. The largest traveling salesman instance solved to date is an instance on 13,509 cities (more than 91 million variables) solved by Applegate, Bixby, Chvátal, and Cook in 1998. For more information on solving the traveling salesman problem I refer to the home page of William Cook: `http://www.caam.rice.edu/~bico/`. Of course, not only traveling salesman instances have been tackled, but since it is probably the most well-known integer programming problem, and since many algorithmic techniques are compared on traveling salesman instances, this has served as an illustration here. The book by for instance Nemhauser and Wolsey, together with the survey paper by Marchand et al., will give a good picture of the developments of cutting plane algorithms during the past decades. What is common for most recent computational successes is that they combine both Gomory cutting planes and facet defining inequalities with branch-and-bound based on the LP-relaxation. Branch-and-bound is an enumerative algorithm that partitions the set of solutions into subsets such that the intersection of all subsets is empty and such that the union of the subsets is the complete set. Instead of solving the integer programming problem on the subset, one solves the LP-relaxation instead. The information from the LP-relaxation is used in order to prune the search tree, i.e., to avoid looking at all possible solutions.

Before closing this section I just want to give a very brief impression of what can, and what cannot be done in terms of solving real-life integer programming problems at present. From the above discussion it is clear that a huge progress has been made in solving traveling salesman instances to optimality. A similar progress has been made for many other problems in which the variables may take values 0 or 1 only. Here large-scale instances arising in for instance frequency assignment in mobile communication, and network design, such as the problem described in Example 1 can be solved. Problems arising in machine scheduling are still very difficult to tackle, which is also the case for many vehicle routing problems. Here one is typically satisfied with a good feasible solution, and the foremost methods for finding such solutions are based on so-called local search techniques. Problems in which the integer variables are not restricted to the values 0 or 1 are also more difficult to tackle than the 0-1 problems. What we have learned over the years is that one needs to study the structure of integer programming problems carefully and use this structure well in algorithms should there be a hope of solving large instances. For some problems it is clear that using the LP-relaxation and cutting planes is not capturing the structure well enough. Some of these problems have been successfully solved by other techniques such as for instance constraint programming,

and lattice basis reduction. I will say something briefly about that in Section 4. For a more thorough overview of "non-standard" techniques for solving integer programming we refer to the survey of Aardal, Weismantel and Wolsey (1999).

3 Complexity and algorithmic issues

A natural first question for a computer scientist studying integer programming is: What is the computational complexity of problem (1)? Karp (1972) proved that the decision version of the 0-1 integer programming problem is NP-complete, and Borosh and Treybig (1976) proved that the decision version of (1) belongs to NP. Combining these results implies that the decision version of (1) is NP-complete. This implies that we cannot expect that there exists a polynomial algorithm for solving (1). Is this a reason to give up? On the contrary! It is a reason to try harder. Here my view may be slightly controversial. I think that computer scientists are often too afraid of "NP-complete" and "NP-hard". As I mentioned in the previous section, finding an explicit description of $\text{conv}(X)$ is hard in general. More precisely, for general polyhedra X we cannot hope for an explicit description of $\text{conv}(X)$ unless NP = co-NP (Karp and Papadimitriou, 1980). But this is not a reason to discard algorithms based on cutting planes. Surprisingly many instances of NP-hard problems can be solved to optimality if one puts a lot of effort into analyzing the structure of the problem and if one is careful with the design and implementation of the algorithm. Part of the issue is that the measure of hardness involves a "worst case", which is not a very refined measure.

If we want to solve an NP-hard problem to optimality, and believe that P$\neq$NP, then we have to resort to an algorithm with superpolynomial time complexity, so a large part of the fun of analyzing it is gone. Therefore, in computer science we often do one of the following things: either we drop the problem altogether, or we develop a so-called polynomial time approximation algorithm with a performance guarantee. Such an algorithm finds, for all instances, a feasible solution with value less than or equal to ρ times the optimal value, where ρ is the performance guarantee. I must stress that I am not at all against the topic of polynomial time approximation algorithms with a performance guarantee. This area has been exceptionally active during the past decade, and many beautiful algorithmic ideas have been developed. The only thing I am slightly critical of is the motivation used by many researchers. In a majority of papers on the subject one can find a version of the following sentence: "the problem is NP-hard and therefore it cannot be solved to optimality within reasonable time". How many of the researchers actually tried it? Another issue is that the problems for which polynomial approximation algorithms with performance guarantee have been developed are typically the integer programming problems that have a very clean structure and these are in many cases (not all!) the problems that we know relatively well how to solve. (Machine scheduling problems are important examples of problems that are still very difficult to solve to optimality.) So, in terms of finding good feasible solutions for difficult integer programming problems we still need to learn more. I think a better motivation for studying polynomial time approximation algorithms is that finding one does answer an interesting mathematical question, that the algorithmic ideas are beautiful, and that we often learn more of the aspect of hardness of a problem even though one again needs to be careful with one's intuition here. One curious problem in this respect is the

uncapacitated facility location problem, in which we wish to decide on where to set up facilities, and from which of the open facilities clients belonging to a given set should be served. All demand of the clients should be satisfied and the total cost of setting up the facilities and of transporting the goods from the facilities to the clients should be minimized. The problem is NP-hard, yet it is quite easy to solve to optimality for most realistic instance sizes. If one makes the assumptions that it is possible to locate a facility at precisely the points in which a client is located, that the fixed costs are positive, and that the transportation costs are nonnegative and symmetric, i.e., going from a to b is as expensive as going from b to a, then one has to work quite hard to even construct an instance for which the LP-relaxation has a non-integer solution! Note that the assumptions made above are not unrealistic in a practical case. The question of finding an approximation algorithm with a constant value of the performance guarantee ρ for the uncapacitated facility location problem was posed in the mid-seventies, and for a long time only a logarithmic worst case guarantee was known, which some people interpreted as a sign that this problem was *really* hard. In 1997 the first polynomial time approximation algorithm with a constant value of ρ was found. The best known performance guarantee at present is due to Chudak (1998) who developed an approximation algorithm with a worst case guarantee of 1.736. It was proved, almost simultaneously, that we cannot expect to find a polynomial time approximation algorithm with a performance guarantee less than 1.463. Again, both these results are beautiful and interesting, but from the computational point of view it is not so clear that a polynomial time approximation algorithm with performance guarantee is the best algorithm to use if the goal is to find good-quality solutions, since it has been designed to "get around" the worst case, and not primarily to perform well on the most typically occurring instances. One detail worth noting in this context is that many polynomial time approximation algorithms make use of information from the LP-relaxation. In the proof of the running time, the LP-relaxation is solved by the ellipsoid algorithm, which is polynomial. It is, however, a well-known fact that the ellipsoid algorithm is not practical to use, so in an implementation one would use the simplex algorithm for which no polynomial time version is known. So, strictly speaking, the resulting implemented algorithm would not be polynomial.

Before turning to the next section I want to mention a few other interesting complexity results. One question that kept intriguing the mathematical programming community for several years was whether or not the *linear* programming problem (2) can be solved in polynomial time. It was shown fairly quickly after complexity theory was developed (Cook, 1971, Karp, 1972) that LP belongs to NP $\cap$ co-NP, so it was natural to think that it was polynomially solvable. Note that membership of LP in co-NP follows from LP-duality. No pivoting rule of the simplex has been proved to yield a polynomial time algorithm. The question was answered in 1979 by Khachiyan who proved that the so-called *ellipsoid algorithm* runs in polynomial time when applied to the linear programming problem. Very rarely has a mathematical result attracted so much publicity (see Lawler, 1980)! With the computational breakthrough of cutting plane techniques the following question arose: What is the relation between the optimization problem (1) and the so-called *separation problem*? The separation problem based on a family of polyhedra $\mathcal{P}$ is formulated as follows: given a vector x^* and a polyhedron $P \in \mathcal{P}$, show that $x^* \in P$ or give an inequality $d^T x \leq d_0$ such that $d^T x^* > d_0$. Grötschel, Lovász and

Schrijver (1981) showed that the optimization problem and the separation problem are polynomially equivalent, i.e., the optimization problem is polynomially solvable if and only if the separation problem is polynomially solvable. Another interesting complexity question is whether the integer programming problem can be solved in polynomial time if the number of variables is fixed. This question was answered in the affirmative by H.W. Lenstra, Jr (1983). It is easy to construct an instance of (1) on which branch-and-bound performs arbitrarily bad, even the number of variables is just two. Lenstra's algorithm was receiving a lot of attention as it introduced concepts from number theory to the theory of integer programming. An important tool of his algorithm was *lattice basis reduction* (Lenstra, Lenstra, Lovász, 1982), which has been used successfully in computational integer programming as well, see the survey by Aardal et al.

4 Discussion

What are the challenging questions in integer programming today? There are many of them, but I can mention only a few here. The most obvious one is maybe: Is P = NP? The other questions I want to mention are related to computational integer programming. Solving real-life integer programming problems still poses a serious challenge. Something that may be viewed as surprising is that local search algorithms often perform well on such problems, i.e., the locally optimal solutions produced by these algorithms, if designed and implemented carefully, are often of very good quality, see e.g. the book edited by Aarts and Lenstra (1997). Why is this the case?

Another interesting question is to find out more about what actually makes a problem type hard. NP-hardness is a quite coarse measure as discussed in the previous sections. We have also seen many examples recently of problem instances that are hard even in very low dimension such as $n = 10$, so hard and large are by no means strictly related. When I say "hard" here I mean that these instances belong to NP-hard problems and that they cannot be solved by cutting planes combined by branch-and-bound as discussed in Section 2, or by any obvious variant of this approach. Some of the known instances of this type have been solved by algorithms based on basis reduction, see Aardal et al., but only some ideas exist as to why these techniques work well here.

What will the future bring in terms of computers and computer models? Will the whole notion of "hard" disappear once we start computing on quantum computers, or use plastic instead of silicon, or DNA-chains? Well, the future will show, but I am convinced that there will always be really hard instances, irrespective of the computer models, so understanding the problem structure will remain important.

References

K. Aardal. 1996. Interview with Ralph E. Gomory. *OPTIMA - Mathematical Programming Society Newsletter* No. 50, 1–3, 5–6.

K. Aardal, R. Weismantel, L.A. Wolsey. 1999. Non-standard approaches to integer programming. Report UU-CS-1999-41, Department of Computer Science, Utrecht University.

E.H.L. Aarts, J.K. Lenstra (eds.). 1997. *Local Search in Combinatorial Optimization.* John Wiley & Sons Ltd., Chichester.

E. Balas, S. Ceria, G. Cornuéjols, N.R. Natraj. 1996. Gomory cuts revisited. *Operations Research Letters* **19** 1–9.

I. Borosh, L.B. Treybig. 1976. Bounds on positive integral solutions of linear diophantine equations. *Proceedings of the American Mathematical Society* **55**, 299–304.

F. Chudak. 1998. Improved approximation algorithms for uncapacitated facility location. *Integer Programming and Combinatorial Optimization: 6th International IPCO Conference.* Lecture Notes in Computer Science **1412** Springer-Verlag, Berlin, Heidelberg, pp 180–194.

S.A. Cook. 1971. The complexity of theorem-proving procedures. *Proceedings of Third Annual ACM Symposium on Theory of Computing*, pp 151–158, ACM, New York.

G.B. Dantzig. 1951. Maximization of a linear function of variables subject to linear inequalities. Tj. C. Koopmans, ed. *Activity Analysis of Production and Allocation*, J. Wiley & Sons Inc., New York, 339–347.

G.B. Dantzig, D.R. Fulkerson, S.M. Johnson. 1954. Solution of a large-scale traveling-salesman problem. *Operations Research* **2** 393–410.

R.E. Gomory. 1958. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society* **64** 275–278.

R.E. Gomory. 1960. Solving linear programming problems in integers. R.E. Bellman, M. Hall, eds., *Combinatorial Analysis.* American Mathematical Society, 211–216.

M. Grötschel. 1980. On the symmetric traveling salesman problem: solution of a 120 city problem. *Mathematical Programming* **12** 61–77.

M. Grötschel, L. Lovász, A. Schrijver. 1981. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica* **1** 169–197. [Corrigendum: **4** (1984) 291–295.]

M. Grötschel, L. Lovász, A. Schrijver. 1988. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, Berlin.

S. Guha, S. Khuller. 1998. Greedy strikes back: Improved facility location algorithms. *Ninth ACM-SIAM Annual Symposium on Discrete Algorithms.*

R.M. Karp. 1972. Reducibility among combinatorial problems. R.E. Miller and J. W. Thatcher, eds., *Complexity of Computer Computations* pp 85–103, Plenum Press, New York.

R.M. Karp, C.H. Papadimitriou. 1980. On linear characterization of combinatorial optimization problems. *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, IEEE, New York, pp 1–9.

L.G. Khachiyan. 1979. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR* **244** 1093–1096. [English translation: *Soviet Mathematics Doklady* **20** (1979) 191–194.]

E.L. Lawler. 1980. The great mathematical sputnik of 1979. *The Sciences* September Issue.

A.K. Lenstra, H.W. Lenstra, Jr., L. Lovász. 1982. Factoring polynomials with rational coefficients. *Mathematische Annalen* **261** 515–534.

H.W. Lenstra, Jr. 1983. Integer programming with a fixed number of variables. *Mathematics of Operations Research* **8** 538–548.

H. Marchand, A. Martin, R. Weismantel, L.A. Wolsey. 1999. Cutting planes in integer and mixed integer programming. Report CORE DP 9953, CORE, Université Catholique de Louvain, Louvain-la-Neuve.

G.L. Nemhauser, L.A. Wolsey. 1988. *Integer and Combinatorial Optimization.* John Wiley & Sons Inc., New York.

A. Schrijver. 1986. *Theory of Linear and Integer Programming.* John Wiley & Sons Ltd., Chichester.

M. Stoer. 1992. *Design of Survivable Networks.* Lecture Notes in Mathematics **1531**, Springer-Verlag, Berlin, Heidelberg.

6.2 Treewidth – a brief introduction: Hans L. Bodlaender

Department of Computer Science, Utrecht University, Padualaan 14, P.O. box 80.089, 3508 TB Utrecht, The Netherlands
e-mail: hansb@cs.uu.nl

Abstract

In this paper, a brief introduction to the algorithmic theory of graphs of bounded treewidth is given.

Introduction

Trees form a well known class of graphs with many interesting properties, also algorithmically. One of these properties is that many problems, that are computationally hard (e.g., NP-complete) for general graphs become polynomial time solvable when restricted to trees: often with some form of dynamic programming. In the past decades, several classes of graphs which have some kind of 'tree-like structure' have been proposed. One of the more popular and successful generalisations of the notion of tree has been the notion of *partial k-tree*, or, equivalently, graphs with bounded treewidth.

Treewidth is a measure of graphs. There are several equivalent definitions of the same notion: the one using tree decompositions is given in Section 2; but one can also define the treewidth of a graph G as the minimum k, such that G is a partial k-tree (subgraph of a k-tree), or, again equivalently, such that G is the subgraph of a chordal graph with chromatic number k. An overview of these, and other equivalent definitions (e.g., in terms of searching games, or with help of graph composition operations) can be found in [9].

Perhaps the most important reason of the interest in the notion of treewidth amongst researchers in algorithmic graph theory was that the fact that many problems are 'easy' for trees usually generalises to graphs of bounded treewidth: many problems that are hard (e.g., NP-hard) for arbitrary graphs become polynomial time solvable (and often linear time solvable) on classes of graphs where a uniform upper bound on the treewidth of the graphs in the class is known. This is discussed in Section 3. There are several general results, showing for all problems in a large class of problems that they allow these type of algorithms. In Section 4, we discuss one such type of general results, based on (monadic second order) logic. In Section 5, results on how to find tree decompositions are briefly discussed, and some final remarks are given in Section 6.

Treewidth and tree decompositions

The notion of treewidth was introduced by Robertson and Seymour in [18].

Definition 1 *A* tree decomposition *of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ with $\{X_i \mid i \in I\}$ a family of subsets of V, one for each node of T, and T a tree such that*

- $\bigcup_{i \in I} X_i = V$.
- *for all edges $(v, w) \in E$, there exists an $i \in I$ with $v \in X_i$ and $w \in X_i$.*
- *for all $i, j, k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*

The width *of a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i \in I} |X_i| - 1$. The* treewidth *of a graph G is the minimum width over all possible tree decompositions of G.*

As discussed earlier, there are several equivalent definitions. It is possible to manipulate a tree decompositions in some ways to get tree decompositions of some special type. For instance, when one has a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of a graph $G = (V, E)$ one can transform it in time, linear in $|I|$, into a tree decomposition $(\{X'_i \mid i \in I'\}, T' = (I', F'))$ of the same width, where T is a binary tree, and $|I'| = O(|I|)$, and $|I'| = O(|V|)$, i.e., the tree is of size, at most linear in size of the tree of the original tree decomposition and linear in the number of vertices of G. (For details and more similar results, see e.g. the overview in [9].)

Solving problems with tree decompositions

An illustrative example of the technique how combinatorial problems can be solved using tree decompositions of bounded width can be given with help of the 3-COLORING PROBLEM: given a graph $G = (V, E)$, can we assign each vertex a color from the set $C = \{\text{red, white, blue}\}$, such that adjacent vertices are given a different color (called a proper 3-coloring below)? This problem is well known to be NP-complete, but when a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of width at most k is given, then the problem can be solved in time $O(3^k \cdot |I|)$. As tree decompositions can have a number of nodes, linear in the number of vertices of the graph, this is a linear time algorithm when k is constant. The algorithm given below has a slightly larger constant factor (about 3^{3k}), but shows the basic principle very well.

As discussed in Section 2, we may assume that T is a binary tree. We choose any node r of degree at most two as root of T. For a node $i \in I$, let $T_i = (I_i, F_i)$ be the subtree of T, rooted at node i. Write $V_i = \bigcup_{j \in I_i} X_i$, i.e., V_i is formed by all vertices in the set X_i or sets X_j, j a descendant of i.

Now, write F_i as the function, that maps each function $c : X_i \to C$ to either true or false, with $F_i(c) =$ true, if and only if there is a coloring $c' : V_i \to C$ with adjacent vertices always different colors, such that for all $v \in X_i$, $c(v) = c'(v)$, or, in other words, c can be extended to a proper 3-coloring of G_i.

Now, for leaves i, the function F_i can easily be tabulated: $F_i(c)$ is true, if and only if c is a proper 3-coloring. Essential is however, that for other nodes i, F_i can be tabulated when the tables, giving these functions for the children of i are known. Suppose i has children j_i and j_2. The definition of tree decomposition makes that a vertex v in G_{j_1} can only be adjacent to a vertex $w \in G_{j_2}$, if $v \in X_i$ or $w \in X_i$. Using this, one can show that for a function $c : X_i \to C$, we have $F_i(c)$, if and only if no two adjacent vertices are given the same color by function c, and there exist functions $c_1 : X_{j_1} \to C$, $c_2 : X_{j_2} \to C$ with $F_{j_1}(c_1) =$ true, $F_{j_2}(c_2) =$ true; and for $v \in X_i \cap X_{j_1}$: $c(v) = c_1(v)$, $v \in X_i \cap X_{j_2}$: $c(v) = c_2(v)$. This property can be checked in $O(1)$ time for each c, given tables for the values of F_{j_1} and F_{j_2}, assuming the width k is a constant.

Thus, we can compute tables for all functions F_i, $i \in I$, in a bottom-up way in the tree: starting at leaves, and computing a table for a node after the tables for its children are done. When we have the table of the root node, we know whether G is 3-colorable: then there is at least one function $c : X_r \to C$ with $F_r(c) =$ true.

Another way of viewing this algorithm is that we compute for each i a set of equivalence classes of *partial solutions*: two proper 3-colorings c_1 and c_2 of G_i are equivalent when for all $v \in X_i$, $c_1(v) = c_2(v)$. If c_1 and c_2 are equivalent, then we can extend c_1 to a proper 3-coloring of G by assigning colors to vertices in $V - V_i$, if and only if c_2 can similarly be extended to a proper 3-coloring of G. Thus, what we do is tabulate all non-empty equivalence classes for each node of the tree decomposition.

This method can be used for many combinatorial problems. (See [8, 6] for introductions with many references.)

One recent example of a successful application of the theory can be found in [15], where Koster uses tree decompositions to solve specific instances of the MINIMUM INTERFERENCE FREQUENCY ASSIGNMENT PROBLEM, a problem arising from the need to assign frequencies to radio transmitters, which can be seen as a generalisation of graph coloring.

Similar algorithms are also used in other fields, e.g., for doing otherwise intractable computations on Bayesian belief networks (see e.g., [19, 17].)

Note that in the algorithm for the 3-COLORING PROBLEM described above, the information that is computed for each node is of constant size. In that case, the dynamic programming algorithm can also be viewed as a *tree automaton*. This is perhaps easier illustrated by first looking at the case that we have path-decompositions (tree decompositions where T has no nodes of degree more than two, i.e., where T is a path). In that case, we compute some information for each node, in order; the information that we compute depends only on what is computed for the previous node, and the structure of node i (e.g., the subgraph of G induced by X_i.) Thus, this algorithm can be seen as a finite state automaton - the structural information of nodes corresponds to the symbols

of the string; the computed information of nodes to the state the machine is in, etc. For trees instead of paths, the situation is similar, but now we have a finite state tree automaton. (See [1].) Using the terminology of Courcelle, we call problems that allow such a 'tree automaton' (dynamic programming algorithm with a constant number of bits per node) *recognisable.*

Monadic second order logic and more

There are several results that state for a large class of combinatorial problems that each of the problems in the class can be solved in linear time, (or polynomial time) when restricted to graphs of bounded treewidth.

Perhaps the most successful of these kinds of results are the results on problems, expressible in *Monadic Second Order Logic*, or an extension of this class. This work was started by Courcelle (see e.g., [13]), and later several authors found extensions of his result. Independently, but perhaps somewhat later, a similar result was found by Borie et. al [11]. See also [4].

A problem is expressible in Monadic Second Order Logic, if it can be formulated using the following language constructions: quantification over vertices, sets of vertices, edges, sets of edges, set membership of vertices, set membership of edges, adjacency tests (are these vertices adjacent, is this vertex incident to this edge?), and logic operations (and, or, etc.) For example, the 3-COLORING PROBLEM can be expressed as follows:

$$\exists V_1 \subseteq V : \exists V_2 \subseteq V : \exists V_3 \subseteq V : (\forall v \in V : v \in V_1 \vee v \in V_2 \vee v \in V3) \wedge (\forall v \in V : \forall w \in V : (v \in V_1 \wedge w \in V_1 \rightarrow \text{not } (v,w) \in E) \wedge (v \in V_2 \wedge w \in V_2 \rightarrow \text{not } (v,w) \in E) \wedge (v \in V_3 \wedge w \in V_3 \rightarrow \text{not } (v,w) \in E))$$

Courcelle [13] showed that every graph problem, expressible in Monadic Second Order Logic can be solved in linear time, when the input graph is given with a tree decomposition of width bounded by a constant.

There are extensions of the language that allow to express optimisation problems which also allow linear time algorithms for graphs of bounded treewidth. E.g., one can allow to ask for the minimum or maximum size set of vertices or edges such that a MSOL-expressible property holds. The MAXIMUM INDEPENDENT SET PROBLEM (find a set of non-adjacent vertices in a given graph G that is as large as possible) can be expressed as follows:

$$\max_{W \subseteq V} |W| : \forall v \in V : \forall w \in V : (v \in W \wedge w \in W) \rightarrow (\text{ not } (v,w) \in E)$$

An important extension of Monadic Second Order Logic is *Counting Monadic Second Order Logic.* Let $m_{r,s}$ be the predicate on sets for which $m_{r,s}(Z) = \text{true}$, if and only if $|Z| \bmod r = s$. Expressions in Counting Monadic Second Order Logic can use all constructions of MSOL, and predicates of the form $m_{r,s}(W)$ or $m_{r,s}(F)$ on vertex set or edge set variables.

Call a property *definable*, when it can be expressed in CMSOL. Courcelle conjectured that a problem is definable, if and only if it is recognisable. This conjecture was recently proved by Lapoire [16].

LaPoire's theorem can be seen as a generalisation of the result by Büchi from 1960, which states that the regular languages are precise the languages definable by a formula in Monadic Second Order Logic [12].

Finding tree decompositions

For the algorithms discussed in the previous two sections, it is required that one has a tree decomposition of the graph given of bounded width. For some special classes of graphs, (e.g., outerplanar graphs), such a tree decomposition can be constructed easily, but in general, this problem is hard. In general, it is NP-hard to determine the treewidth of a given graph [2], but many special cases can be solved in polynomial time. (Overviews of several such results can be found in [8, 6].)

An important case is the 'fixed parameter case': for constants k, there is a linear time algorithm that given a graph, determines if the graph has treewidth at most k, and if so, finds a tree

decomposition of width at most k [7]. The algorithm however suffers from a high constant factor. Few experimental work has so far been done on finding 'good tree decompositions', but see e.g. [5]. A good introduction to fixed parameter complexity is [14].

Final remarks

It is also possible to solve problems on graphs of bounded treewidth by graph reduction techniques. Arnborg et al. [3] showed that all recognisable problems can be solved in linear time (but more than linear space) on graphs of bounded treewidth using reduction. There is no need to first find a tree decomposition. The algorithm works by repeatedly modifying the graph into a smaller, equivalent graph. These results were generalised in [10].

Several aspects of the algorithmic work on graphs of bounded treewidth were not discussed in this short paper. For instance, a rather interesting result is that of Robertson and Seymour, whose deep results on graph minors imply that every minor closed class of graphs has a polynomial time recognition algorithm; if in addition, the class does not contain all planar graphs, then tree decompositions can be used to (non-constructively) show the existence of a linear time recognition algorithm. (See e.g. [14].)

References

[1] ABRAHAMSON, K. R., AND FELLOWS, M. R. Finite automata, bounded treewidth and well-quasiordering. In *Proceedings of the AMS Summer Workshop on Graph Minors, Graph Structure Theory, Contemporary Mathematics vol. 147* (1993), N. Robertson and P. Seymour, Eds., American Mathematical Society, pp. 539–564.

[2] ARNBORG, S., CORNEIL, D. G., AND PROSKUROWSKI, A. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth. 8* (1987), 277–284.

[3] ARNBORG, S., COURCELLE, B., PROSKUROWSKI, A., AND SEESE, D. An algebraic theory of graph reduction. *J. ACM 40* (1993), 1134–1164.

[4] ARNBORG, S., LAGERGREN, J., AND SEESE, D. Easy problems for tree-decomposable graphs. *J. Algorithms 12* (1991), 308–340.

[5] BECKER, A., AND GEIGER, D. A sufficiently fast algorithm for finding close to optimal junction trees. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence* (1996), Morgan Kaufmann, pp. 81–96.

[6] BODLAENDER, H. L. A tourist guide through treewidth. *Acta Cybernetica 11* (1993), 1–23.

[7] BODLAENDER, H. L. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput. 25* (1996), 1305–1317.

[8] BODLAENDER, H. L. Treewidth: Algorithmic techniques and results. In *Proceedings 22nd International Symposium on Mathematical Foundations of Computer Science, MFCS'97, Lecture Notes in Computer Science, volume 1295* (Berlin, 1997), I. Privara and P. Ruzicka, Eds., Springer-Verlag, pp. 19–36.

[9] BODLAENDER, H. L. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comp. Sc. 209* (1998), 1–45.

[10] BODLAENDER, H. L., AND DE FLUITER, B. Reduction algorithms for constructing solutions in graphs with small treewidth. In *Proceedings 2nd Annual International Conference on Computing and Combinatorics, COCOON'96* (1996), J.-Y. Cai and C. K. Wong, Eds., Springer Verlag, Lecture Notes in Computer Science, vol. 1090, pp. 199–208.

[11] BORIE, R. B., PARKER, R. G., AND TOVEY, C. A. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica 7* (1992), 555–581.

[12] BÜCHI, J. R. Weak second order logic and finite automata. *S. Math. Grundlagen Math. 5* (1960), 66–92.

[13] COURCELLE, B. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation 85* (1990), 12–75.

[14] DOWNEY, R. G., AND FELLOWS, M. R. *Parameterized Complexity.* Springer, 1998.

[15] KOSTER, A. M. C. A. *Frequency assignment - Models and Algorithms.* PhD thesis, Univ. Maastricht, Maastricht, the Netherlands, 1999.

[16] LAPOIRE, D. Recognizability equals definability, for every set of graphs of bounded tree-width. In *Proceedings 15th Annual Symposium on Theoretical Aspects of Computer Science* (1998), Springer Verlag, Lecture Notes in Computer Science, vol. 1373, pp. 618–628.

[17] LAURITZEN, S. J., AND SPIEGELHALTER, D. J. Local computations with probabilities on graphical structures and their application to expert systems. *The Journal of the Royal Statistical Society. Series B (Methodological) 50* (1988), 157–224.

[18] ROBERTSON, N., AND SEYMOUR, P. D. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms 7* (1986), 309–322.

[19] VAN DER GAAG, L. C. *Probability-Based Models for Plausible Reasoning.* PhD thesis, University of Amsterdam, 1990.

6.3 Speaking of objects: F.S. de Boer

Utrecht University, The Netherlands
Email: frankb@cs.uu.nl

Abstract

This note discusses an assertion language for describing properties of configurations of objects and its application to reasoning about aliasing and object creation.

Introduction

The execution of an object-oriented program gives rise to a dynamic configuration of objects. Characteristic of such a configuration is that objects can be created at arbitrary points during the execution of a program, and references to objects, 'pointers', can be stored in variables and passed around, for example, as parameters in messages. This implies that complicated and dynamically evolving structures of references between objects can occur.

The problem addressed in this note is an appropriate assertion language for describing properties of these dynamic configurations. An important requirement is that the *abstraction level* of the assertion language should coincide with that of the programming language. In more detail, this means the following:

- The only operations on "pointers" (references to objects) are testing for equality and dereferencing (looking at the value of an instance variable of the referenced object).
- Furthermore, in a given configuration, it is only possible to mention the objects that exist in that configuration. Objects that do not (yet) exist never play a role.

A consequence of the limited set of operations on pointers is that first-order logic is too weak to express some interesting properties of pointer structures (for example, the property, as considered in [6], that it is possible to go from one object to the other by following a finite number of x-links). Therefore we have to extend our assertion language to make it more expressive. In this note this is done by allowing the assertion language to reason about finite sequences of objects (in programming jargon: arrays of objects).

This note is organized as follows. In the following section the assertion language is introduced. In section 3 it is shown how to model assignments which involve aliasing and object-creation logically by substitutions which, when applied to a given postcondition, yield the corresponding weakest precondition.

The assertion language

In this section a formalism is introduced for expressing certain properties of a configuration of objects. An object itself is an entity containing data and procedures (*methods*) acting on these data. The data are stored in *variables*, which come in two kinds: *instance variables*, whose lifetime is the same as that of the object they belong to, and *temporary variables*, which are local to a method and last as long as the method is active. Variables can contain references to other objects in the system (or even the object under consideration itself). The value of a variable can be changed by an *assignment*.

Without loss of generality, we may assume that the variables of an object cannot be accessed directly by other objects. The only way for objects to interact is by sending *messages* to each other. If an object sends a message, it specifies the receiver by means of one of its variables, a method name, and possibly some parameters. Then control is transferred from the sender object to the receiver. This receiver then executes the specified method, using the parameters in the message. Note that this method can, of course, access the instance variables of the receiver. The method returns a result which is sent back to the sender. Then control is transferred back to the sender which resumes its activities, possibly using this result.

Given an arbitrary (infinite) set O of ***object identities***, with typical element o, a configuration of objects can be formally defined as a ***partial*** function σ on O such that for every $o \in O$, $\sigma(o)$, if defined, denotes the internal state of object o. The internal state of an object assigns a value to each of its variables. This value may be of some predefined data type like that of the integers or booleans, or it is an element of the set O of object identities. An object $o \in O$ ***exists*** in σ if $\sigma(o)$ is defined. We restrict to configurations which are ***consistent*** in the following sense. The variables of existing objects refer to existing objects. Formally, a configuration σ is consistent if for every existing object o, $\sigma(o)(x) \in O$ implies that $\sigma(o)(x)$ is an element of the domain of σ.

We assume given an object-oriented programming language with the following typical elements of the set of expressions without ***side-effects*** (we omit the typing information).

$$e ::= \mathsf{self} \mid \mathsf{nil} \mid n \mid \mathsf{true} \mid \mathsf{false} \mid x \mid e_1 + e_2 \mid \cdots$$

Such an expression e is evaluated by an object o in a configuration of objects σ. The result of this evaluation is denoted by $Val(e)(o, \sigma)$, with o an object existing in σ. It is formally defined by a simple induction on the structure of e. For example, $Val(\mathsf{self})(o, \sigma) = o$, i.e., the expression self denotes the object o itself, and $Val(x)(o, \sigma) = \sigma(o)(x)$. In the grammar above, n denotes an integer constant, the arithmetical operation of addition is denoted by $+$, and true and false denote the standard boolean values. Finally, the expression nil stands for 'undefined'.

One element of the assertion language for describing properties of configurations of objects will be the introduction of ***logical variables***. These variables may not occur in the program, but only in the assertion language. Therefore we are always sure that the value of a logical variable can never be changed by a statement. Logical variables are used to express the constancy of certain expressions (for example in a proof rule for message passing, see [4]). Logical variables also serve as bound variables for quantifiers.

In general, the set of expressions in the assertion language will be larger than the set of programming language expressions not only because it contains logical variables, but also because we include conditional expressions in the assertion language. These conditional expressions will be used for the analysis of the phenomenon of aliasing which arises because of the presence of a dereferencing operator.

In two respects the assertion language differs from the usual first-order predicate logic: Firstly, the range of quantifiers is limited to the *existing* objects in the configuration under consideration. For the classes different from the predefined ones like that of the integers and booleans this restriction means that we cannot talk about objects that have not yet been created, even if they could be created in the future. This is done in order to satisfy the requirements stated in the introduction. Because of this the range of the quantifiers can be different for different states. More in particular, a programming statement can change the truth of an assertion even if none of the program variables accessed by the statement occurs in the assertion, simply by creating an object and thereby changing the range of a quantifier. (The idea of restricting the range of quantifiers was inspired by [8].)

Secondly, in order to strengthen the expressiveness of the logic, it is augmented with quantification over finite sequences of objects. It is quite clear that this is necessary, because simple first-order logic is not able to express certain interesting properties.

We give the following typical elements of the set logical expressions (we omit the typing information):

$$l ::= e \mid z \mid l.x \mid \mathsf{if}\ l_0\ \mathsf{then}\ l_1\ \mathsf{else}\ l_2\ \mathsf{fi}$$

As described above, the expression e comes from the given programming language. It is evaluated by the object which is denoted by self. The variable z is a logical variable. We can now also look at the value of an instance variable of the referenced object: the expression $l.x$ refers to the local value of the *instance* variable x of the object denoted by l. So we assume that x in this context is an instance variable because temporary variables here do not make sense. Moreover, as explained above, we can write conditional expressions.

Formally, an expression l is evaluated by an object o in a configuration of objects σ and a logical environment ω which assigns values to the logical variables. The result of this evaluation is

denoted by $Val(l)(\omega, o, \sigma)$. It is defined by a simple induction on the structure of l. For example, $Val(z)(\omega, o, \sigma) = \omega(z)$ and $Val(l.x)(\omega, o, \sigma) = \sigma(o')(x)$, where $o' = Val(l)(\omega, o, \sigma)$.

In order to reason about sequences we assume the presence of notations to express, for example, the length of a sequence (denoted by $|l|$) and the selection of an element of a sequence (denoted by $l(n)$, where n is an integer expression).

The set of assertions, with typical element P, is defined by:

$$P ::= l \mid P \wedge Q \mid \neg P \mid \exists z P$$

Here l denotes a boolean expression.

As already explained above, a formula $\exists z P$, with z ranging over objects, states that P holds for some *existing* object. A formula $\exists z P$, with z of a sequence type, states the existence of a sequence of existing objects.

Formally, an assertion P is evaluated by an object o in a configuration σ and a logical environment ω. We denote by $o, \sigma, \omega \models P$ that P is true when evaluated by the object o in the configuration σ and the logical environment ω. It is defined by a straightforward induction on the structure of P. For example, we have that $o, \sigma, \omega \models \exists z P$ if and only if there exists an element o' in the domain of σ such that $o, \sigma, \omega\{o'/z\} \models P$ (assuming that z ranges over objects). Here the logical environment $\omega\{o'/z\}$ results from ω by assigning o' to z.

As an example, the following formula states that the object denoted by the logical variable v can be reached from the object denoted by the logical variable u by a finite sequence of objects which are linked by the instance variable x.

$$\exists z \Big(z(1) = u \wedge z(|z|) = v \wedge \forall n \; (0 < n \wedge n < |z| \rightarrow z(n).x = z(n+1)) \Big)$$

Aliasing and object-creation

In this section we show how we can model aliasing and object-creation in the assertion language by means of substitutions.

Aliasing

Let x be an *instance* variable of some object and e be an expression without side-effect (of a given object-oriented programming language). The execution of an assignment $x := e$ by an object consists of assigning the value of e to its variable x. The corresponding substitution operation $[e/x]$ has to account for possible aliases of the variable x, namely, expressions of the form $l.x$: it is possible that, after substitution, l refers to the object itself (i.e. the object denoted by **self**), so that $l.x$ is the same variable as x and should be substituted by e. It is also possible that, after substitution, l does not refer to the object itself, and in this case no substitution should take place. Since we cannot decide between these possibilities by the form of the expression only, a conditional expression is constructed which decides "dynamically". Note that an assignment $x := e$, with x a *temporary* variable, does *not* give rise to aliasing (because no other object can access x in the assertion language). Consequently, in this case the usual notion of substitution suffices.

We have the following main cases of the substitution operation $[e/x]$:

$$\begin{aligned} (l.x)[e/x] &\equiv l[e/x] = \textsf{self then } e \textsf{ else } l[e/x].x \textsf{ fi} \\ (l.y)[e/x] &\equiv (l[e/x]).y \end{aligned}$$

(Syntactic identity is denoted by $\equiv$.) In the second case it is assumed that x and y are distinct instance variables. The definition is extended to assertions other than logical expressions in the standard way.

As a simple example, applying the substitution $[1/x]$ to the assertion $y.x = 0$ results in the assertion if $y = \textsf{self then } 1 \textsf{ else } y.x \textsf{ fi} = 0$. This assertion clearly is logically equivalent to $y \neq \textsf{self} \wedge y.x = 0$.

The following theorem states that $P[e/x]$ is in fact the ***weakest precondition*** of the assertion P (with respect to the assignment $x := e$).

Theorem 2 *We have that*

$$o, \sigma, \omega \models P[e/x] \text{ if and only if } o, \sigma', \omega \models P,$$

where σ' results from σ by assigning to the variable x of object o the value $Val(e)(o, \sigma)$, i.e., $\sigma'(o)(x) = Val(e)(o, \sigma)$ (and in all other cases σ agrees with σ').

Object creation

Next we consider the creation of objects. We want to define the substitution $[\mathsf{new}/x]$ which models the creation of a new object referred to by x. This substitution should model logically the assignment $x := \mathsf{new}$. Execution of an assignment $x := \mathsf{new}$ consists of the creation of a new object and assigning a reference to this object to x.

As with the usual notions of substitution we want the expression after substitution to have the same meaning before the assignment as the unsubstituted expression has after the assignment. However, in the case of the creation of a new object, there are expressions for which this is not possible, because they refer to the new object and there is no expression that could refer to that object before its creation, because it does not exist yet. Therefore the result of the substitution must be left undefined in some cases.

However we *are* able to carry out the substitution in case of assertions, assuming, without loss of expressiveness, that in the assertion language the operations on sequences are limited to $|l|$, i.e. the length of the sequence l, and $l(n)$, i.e. the operation which yields the nth element of l. The idea behind this is that in an assertion a *temporary* variable x referring to the new object can essentially occur only in a context where either one of its instance variables is referenced, or it is compared for equality with another expression. In both of these cases we can predict the outcome without having to refer to the new object. The case of an assignment $x := \mathsf{new}$, with x an *instance* variable, then can be logically modeled by the composition of the substitutions $[y/x]$ and $[\mathsf{new}/y]$, where y is some (fresh) temporary variable. Note that the substitution $[y/x]$ as defined above renames all possible aliases of the variable x into y.

Here are the main cases of the formal definition of the substitution $[\mathsf{new}/x]$, with x a temporary variable, for logical expressions. As already explained above the result of the substitution $[\mathsf{new}/x]$ is undefined for the expression x. Since the (instance) variables of a newly created object are initialized to nil we have

$$(x.y)[\mathsf{new}/x] \equiv \mathsf{nil}$$

If neither l_1 nor l_2 is x or a conditional expression they cannot refer to the newly created object and we have

$$\Big(l_1 = l_2\Big)[\mathsf{new}/x] \equiv \Big(l_1[\mathsf{new}/x]\Big) = \Big(l_2[\mathsf{new}/x]\Big)$$

If either l_1 is x and l_2 is neither x nor a conditional expression (or vice versa) we have that after the substitution operation l_1 and l_2 cannot denote the same object (because one of them refers to the newly created object while the other one refers to an already existing object):

$$\Big(l_1 = l_2\Big)[\mathsf{new}/x] \equiv \mathsf{false}$$

On the other hand if both the expressions l_1 and l_2 equal x we obviously have

$$\Big(l_1 = l_2\Big)[\mathsf{new}/x] \equiv \mathsf{true}$$

We have that $l[\mathsf{new}/x]$ is defined for boolean expressions l.

Let z be a logical variable ranging over objects. The changing scope of a bound occurrence of the variable z induced by the creation of a new object is captured as follows.

$$(\exists z\ P)[\mathsf{new}/x] \equiv (\exists z(P[\mathsf{new}/x])) \vee (P[x/z][\mathsf{new}/x]).$$

The idea of the application of $[\mathsf{new}/x]$ to $(\exists z\ P)$ is that the first disjunct $\exists z(P[\mathsf{new}/x])$ represents the case that P holds for an 'old' object (i.e. which exists already before the creation of the new

object) whereas the second disjunct $P[x/z][\mathsf{new}/x]$ represents the case that the new object itself satisfies P. Since a logical variable does not have aliases, the substitution $[x/z]$ consists of simply replacing every occurrence of z by x. Similarly, it is easy to derive the following clause for universal quantification.

$$(\forall z\ P)[\mathsf{new}/x] \ \equiv\ (\forall z(P[\mathsf{new}/x])) \wedge (P[x/z][\mathsf{new}/x]).$$

As a simple exampe, we have

$$\begin{array}{ll} \Big(\forall z(x = z \vee \mathsf{self} = z)\Big)[\mathsf{new}/x] & \equiv \\ \forall z\Big((x = z \vee \mathsf{self} = z)[\mathsf{new}/x]\Big) \wedge (x = x \vee \mathsf{self} = x)[\mathsf{new}/x] & \equiv \\ \forall z(\mathsf{false} \vee \mathsf{self} = z) \wedge (\mathsf{true} \vee \mathsf{false}) & \end{array}$$

where the last assertion obviously reduces to $\forall z(\mathsf{self} = z)$. This assertion states that self is the only object which exists, which indeed is the weakest precondition of the assertion $\forall z(x = z \vee \mathsf{self} = z)$ with respect to $x := \mathsf{new}$.

Next we consider the case of an occurrence of a bound variable z which ranges over *sequences* of objects. Let z' be a (fresh) logical variable ranging over sequences of boolean values. The variables z and z' together will code a sequence of objects possibly including the newly created object: at the places where z' yields true the value of the coded sequence is the newly created object. Where z' yields false the value of the coded sequence is the same as the value of z. This encoding is described by the substitution operation $[z', x/z]$, the main characteristic cases of which are:

$$\begin{array}{ll} z[z', u/z] & \text{is undefined} \\ \Big(|z|\Big)[z', u/z] & \equiv |z| \\ \Big(z(l)\Big)[z', x/z] & \equiv \mathsf{if}\ z'(l')\ \mathsf{then}\ x\ \mathsf{else}\ z(l')\ \mathsf{fi}, \quad \text{where } l' = l[z', x/z]. \end{array}$$

This substitution operation $[z', x/z]$ is defined for boolean expressions.

Given this encoding we can now define

$$(\exists z\ P)[\mathsf{new}/x] \ \equiv\ \exists z\ \exists z'\ (|z| = |z'| \wedge (P[z', x/z][\mathsf{new}/x]))$$

As an example, consider the following assertion

$$\exists z\Big(|z| = n \wedge \forall u \exists i\ (z(i) = u)\Big),$$

where z ranges over sequences of objects and u ranges over objects themselves. This assertion states that there exist at most n objects. An application of the substitution $[z', x/z]$ to the assertion $|z| = n \wedge \forall u \exists i\ (z(i) = u)$ results in the assertion

$$|z| = n \wedge \forall u \exists i\ \Big(\mathsf{if}\ z'(i)\ \mathsf{then}\ x\ \mathsf{else}\ z(i)\ \mathsf{fi} = u\Big).$$

In order to apply the substitution $[\mathsf{new}/x]$ to this assertion we first eliminate the conditional expression. We obtain

$$|z| = n \wedge \forall u \exists i\ \Big(z'(i) \rightarrow x = u \wedge \neg z'(i) \rightarrow z(i) = u\Big)$$

(assuming that $\neg, \rightarrow, \wedge$ lists these operators in decreasing binding priority). An application of $[\mathsf{new}/x]$ to this latter assertion results in the following:

$$|z| = n \wedge \forall u \exists i\ \Big(z'(i) \rightarrow \mathsf{false} \wedge \neg z'(i) \rightarrow z(i) = u\Big) \wedge \exists i\Big(z'(i) \rightarrow \mathsf{true} \wedge \neg z'(i) \rightarrow \mathsf{false}\Big)$$

This assertion is clearly logically equivalent to the assertion

$$|z| = n \wedge \forall u \exists i\ \Big(\neg z'(i) \wedge z(i) = u\Big) \wedge \exists i\, z'(i)$$

Summarizing the above we obtain as final result the assertion

$$\exists z \exists z' \Big(|z| = |z'| \wedge |z| = n \wedge \forall u \exists i \; (\neg z'(i) \wedge z(i) = u) \wedge \exists i \, z'(i) \Big)$$

This latter assertion clearly is logically equivalent to the assertion

$$\exists z \Big(|z| = n - 1 \wedge \forall u \exists i \; (z(i) = u) \Big)$$

which indeed corresponds with our intuition of the weakest precondition of the assertion which states that there exists at most n objects after the creation of a new object.

The following theorem states that $P[\mathsf{new}/x]$ in general calculates the weakest precondition of P (with respect to the assignment $x := \mathsf{new}$).

Theorem 3 *We have*

$$o, \sigma, \omega \models P[\mathsf{new}/x] \text{ if and only if } o, \sigma', \omega \models P,$$

where σ' results from σ by extending the domain of σ with a new object o', initializing its variables to nil, *and assigning o' to the variable x of the object o.*

Conclusion

The basic ideas underlying the assertion language discussed in this note have been applied in various Hoare logics for reasoning about the correctness of programming constructs in an object-oriented context, like message passing ([4]), parallelism ([2]), and, synchronous ([1, 3]) and asynchronous ([5]) communication between objects.

Moreover it is expected that this assertion language also provides an appropriate basis for specifying such high-level object-oriented programming mechanisms like subtyping, abstract types and inheritance.

Furure work concerns the use of assertions as annotations of object-oriented programs and the construction of a compiler which translates these annotations into corresponding verification conditions which then can be checked semi-automatically by a proof-checker like PVS ([7]).

Acknowledgement This note summarizes joint work with P. America.

References

[1] P. America and F.S. de Boer: Reasoning about dynamically evolving process structures. Formal Aspects of Computing. Vol. 6, No. 3, 1994.

[2] F.S. de Boer: A proof system for the parallel object-oriented language POOL. Proceedings of the seventeenth International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science, Vol. 443, Warwick, England, 1990.

[3] F.S. de Boer: A compositional proof system for dynamic process creation. Proceedings of the sixth annual IEEE symposium on Logics in Computer Science (LICS), IEEE Computer Society Press, Amsterdam, The Netherlands, 1991.

[4] F.S. de Boer: A WP-calculus for OO. Proceedings of Foundations of Software Science and Computation Structures, FOSSACS'99, Lecture Notes in Computer Science, Vol. 1578, 1999.

[5] F.S. de Boer: Reasoning about asynchronous communication in dynamically evolving object structures. Theoretical Computer Science, 1999. Accepted for publication.

[6] J.M. Morris: Assignment and linked data structures. Manfred Broy, Gunther Schmidt (eds.): Theoretical Foundations of Programming Methodology. Reidel, 1982, pp. 35–41.

[7] S. Owre, J. Rushby and N. Shankar: PVS: A Prototype Verification System. Proceedings of the 1th Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, Vol. 607, Springer-Verlag, pp. 748–752, 1992.

[8] D. S. Scott: Identity and existence in intuitionistic logic. M.P. Fourman, C.J. Mulvey, D.S. Scott (eds.): Applications of Sheaves. Proceedings, Durham 1977, Springer-Verlag, 1979, pp. 660–696 (Lecture Notes in Mathematics 753).

6.4 Stochastic Process Algebras: Linking Process Descriptions with Performance: Ed Brinksma, Pedro R. D'Argenio, Joost-Pieter Katoen

Faculty of Computer Science, University of Twente, Enschede

Stochastic Process Algebras: Linking Process Descriptions with Performance

Ed Brinksma, Pedro R. D'Argenio, Joost-Pieter Katoen
Faculty of Computer Science, University of Twente,
P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract

We present a process algebra that allows one to specify concurrent systems with stochastic functionality in a compositional way. The semantics of this process algebra is given in terms of stochastic automata, an extension of automata with clocks that are basically random variables of a continuous or discrete nature. We show that the well-known performance model of generalised semi-Markov processes (GSMPs) defines a proper subset of stochastic automata and give some examples that show the potential benefits of our approach.

Keywords: *GSMP, process algebra, discrete-event simulation, compositionality, equational reasoning*

1 Introduction

In discrete-event simulation the behaviour of a system as it evolves in time is described in terms of a simulation model. Simulation modelling is usually a time-consuming task and is mainly based on human ingenuity and experience. In addition, the difficulty of the design of a simulation model rapidly grows with the increasing magnitude and complexity of the system itself. To ease the description of simulation models, dedicated simulation languages have been developed, such as Demos, GPSS and Simscript, most of which are process-oriented.

State changes in discrete-event simulation models take place at discrete points in time, whereas the time of occurrence of activities is controlled stochastically, i.e. by means of random variables. A mathematical framework for the study of these models—known as stochastic discrete-event systems—is given by Glynn [8]. He presents a *generalised semi-Markov process* (GSMP) theory for such systems.

In this paper we describe a high-level specification language for stochastic discrete-event systems, in particular GSMPs. Our specification language is based on *process algebra*. In process algebras, like ACP, CCS, CSP and LOTOS, a concurrent system is syntactically represented using powerful composition operators which facilitate the development of well-structured specifications. The algebraic nature of the language allows to reason about specifications in an equational way, thus allowing transformation and verification. Traditionally, process algebras focussed on the specification and analysis of qualitative system properties, but in the last decade the interest in extensions with quantitative information has grown significantly. This integration facilitates the analysis of qualitative and quantitative properties in a single framework.

We use an extension of process algebra in which the time of occurrence of actions, the most primitive notions of activity in process algebra, is determined by a continuous or discrete probability distribution of arbitrary nature. To give a formal semantics to our language SPADES (*Stochastic Process Algebra for Discrete-Event Simulation*, symbolised by ♤) we introduce the concept of *stochastic automata*, an extension of labelled automata with clocks, that can be seen as a stochastic variant of timed automata by Alur & Dill [1]. We argue that GSMPs are a subclass of stochastic automata and give a stochastic extension of the expansion law known from (untimed)

process algebra. This law is of central importance for the verification and correctness-preserving transformation of expressions in $\Diamond$ and, thus of GSMPs.

2 Motivation

A simple queueing system. Consider a queueing system in which jobs arrive and wait until they are executed by a single server. An infinite population of jobs is assumed. Jobs arrive with an inter-arrival time that is determined by a continuous probability distribution F while the delay between the processing of two successive jobs is controlled by distribution H. This system is known as a $G/G/1/\infty$-queue, where G stands for general distribution of the arrival and service process, respectively, 1 indicates the number of servers, and ∞ denotes the buffer capacity.

A GSMP description. A typical GSMP description of such queueing system is defined in the following way. The basic ingredients are states and events. To each state z a (non-empty) set of active events $E(z)$ is associated denoting the set of events that can cause transitions out of z. In our example, we let the state space be $\mathbb{N} \times \{ 0, 1 \}$, where the first component of a state indicates the number of jobs that are currently in the system, and the second component indicates the system status (1 = 'a job just arrived', and 0 = 'a job has just been processed'). Initial state is $(0, 1)$. In each state, possible events are the arrival of a job (denoted a) and the completion of a job (denoted c). In the initial state no job completion is possible. Thus, $E(i, j) = \{ a_{i+1}, c_{i+1} \}$ and $E(0, 1) = \{ a_1 \}$. The arrival of a job causes a transition from state $(i, 0)$ or from $(i, 1)$ to state $(i{+}1, 1)$. Completion of a job leads to a transition from $(i{+}1, 0)$ or $(i{+}1, 1)$ to state $(i, 0)$.

To each event $e \in E(z)$ a clock c_e is associated that indicates the amount of time until expiration. Clocks are initialised by probability distribution functions and run backwards. In each state the active event e^* with minimal clock value is selected for execution. The values of the events in z', the successor state of z' on executing e^*, are determined as follows. Clocks of events in $E(z) - \{ e^* \}$ are decreased by the value of c_{e^*}. The clock of any newly active event e in $E(z') - (E(z) - \{ e^* \})$ is set according to the clock-setting distribution $F(c_e)$. All other clocks in z' equal ∞. Due to the conditions imposed on clock-setting distributions, the event e^* in GSMPs is always uniquely determined [8].

In our example, clocks are initialised as follows. On entering state $(i, 1)$ the clock of the next arrival a_{i+1} is initialised according to distribution F, the job inter-arrival time. On entering state $(i, 0)$ with $i \neq 0$ the clock of the next possible job completion c_{i+1} is initialised according to distribution H, the service delay. The clock for c_1 is set in the same way in state $(1, 1)$.

A compositional approach. Although using this description the dynamics of our example GSMP can be determined, it is in absence of any further explanation not easy to understand. This is basically due to the fact that the individual system components, like arrival and server processes, are hard to recognise from the overall system structure. This problem becomes more apparent if we consider GSMPs modelling systems of more realistic magnitude. We say that the specification lacks *compositionality*. The idea that we shall pursue here is to specify GSMPs in a compositional way.

In *process algebra* the specification of our queueing system can be obtained in a hierarchical manner, starting from the specifications of the individual components. If we let $a; p$ denote a process that immediately can perform an action a and then behaves like process p, and $p + q$ denote the process that behaves either like p or like q, then a buffer of infinite capacity can be specified by the set of processes:

$$\begin{aligned} Queue_0 &= a; Queue_1 \\ Queue_{i+1} &= a; Queue_{i+2} + b; Queue_i \text{ for } i \geqslant 0 \end{aligned}$$

where the index indicates the number of jobs in the buffer. Similarly to GSMPs, clocks can be used to model probabilistic delays. Let C be a finite set of clocks. Using the primitives $C \mapsto p$, the process that after expiration of all clocks in C behaves like p, and $\{\!|C|\!\}p$, the process that behaves

like p after any clock x in C is initialised according to some indicated distribution, we obtain for the arrival and server processes:

$$\begin{aligned} \mathit{Arrival} &= \{\!|x_F|\!\}\{x_F\} \mapsto a; \mathit{Arrival} \\ \mathit{Server} &= b; \{\!|y_H|\!\}\{y_H\} \mapsto c; \mathit{Server} \end{aligned}$$

In the *Arrival* process clock x is initialised and starts counting down. Once it has reached the value 0, it expires and action a is enabled. The overall system is described by:

$$\mathit{System} = (\mathit{Arrival} \,||_{\varnothing}\, \mathit{Server}) \,||_{\{a,b\}}\, \mathit{Queue}_0$$

Here, $||$ stands for parallel composition. In process $p\,||_A\,q$, where A is a set of actions, p and q perform actions autonomously, but actions in A should be performed by both. The resulting specification of the $G/G/1/\infty$ system closely resembles the structure of the system itself, is easy to understand, and readily modifiable (for instance, to a queue with finite capacity, or a system in which the service rate depends on the number of waiting jobs).

The formal meaning of a process algebra term is defined in a mathematical model. By defining an appropriate *equivalence* relation on this model one is able to formally compare and transform (e.g. simplify) specifications. If, in addition, this relation is a *congruence*, then such transformation can be carried out component-wise. An equivalence relation is a congruence if for any term a sub-term may be replaced by an equivalent sub-term and an equivalent term results. This compositional transformation reduces the complexity significantly. Finally, due to the algebraic nature of the formalism it is possible to define equational rules on the syntax that allow to perform transformation and simplification at a purely syntactical level, without any reasoning in semantical terms.

3 Stochastic automata and GSMPs

Stochastic automata. The semantics of our process algebra is defined in terms of stochastic automata. (For a formal interpretation of such automata in terms of measure theory, see [6].) This model is strongly related to GSMPs and incorporates, apart from the necessary ingredients to model GSMPs, the possibility of specifying non-determinism. Non-determinism appears if two transitions become enabled simultaneously. This concept is usually absent in stochastic discrete-event systems, but has been widely accepted in the computer science community for the purpose of under-specification in a step-wise design methodology [13]. For simulation purposes, the non-determinism can be resolved using so-called adversaries that schedule the different branches according to some discrete probability distribution [17]. In this way, a mechanism is obtained similar to the probabilistic branching in GSMPs (see also later on).

Definition 1. A *stochastic automaton* is a tuple $(\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \longrightarrow, \kappa, F)$ where: $\mathcal{S}$ is a non-empty set of *locations* with $s_0 \in \mathcal{S}$ being the *initial location*, $\mathcal{C}$ is a set of *clocks*, $\mathbf{A}$ is a set of *actions*, $\longrightarrow \subseteq \mathcal{S} \times (\mathbf{A} \times \wp_{\mathrm{fin}}(\mathcal{C})) \times \mathcal{S}$ is the set of *edges*, $\kappa : \mathcal{S} \to \wp_{\mathrm{fin}}(\mathcal{C})$ is the *clock-setting function*, and $F : \mathcal{C} \to (\mathbb{R} \to [0,1])$ is the *clock-distribution function* such that $F(x)(t) = 0$ for $t < 0$.

We denote $(s, a, C, s') \in \longrightarrow$ by $s \xrightarrow{a,C} s'$, use x and y to denote clocks, and abbreviate $F(x)$ by F_x. To each location s a finite set of clocks $\kappa(s)$ is associated. As soon as location s is entered any clock x in this set is initialised according to its probability distribution function F_x. Once initialised, the clocks start counting down, all with the same rate. A clock expires if it has reached the value 0. The occurrence of an action is controlled by the expiration of clocks. Thus, whenever $s \xrightarrow{a,C} s'$ and the system is in location s, action a can happen as soon as all clocks in the set C have expired. The next location will then be s'.

Example 2. The stochastic automaton that corresponds to the $G/G/1/\infty$ queue from Section 2 is depicted in Figure 1. Here, we represent a location s as a circle containing the clocks that are to be set in s, and denote edges by arrows. The initial location is represented by a circle equipped

with a small ingoing arrow (leftmost circle in second row). Notice that after an a-action always a location is reached in which clock x is set (according to distribution F), and after a c-action always clock y is set (apart from the first location in the upper row) according to H. Clock x thus controls the job inter-arrival time while y controls the service delay. The locations in the upper row represent the states $(i, 0)$ whereas the lower row represents the states $(i, 1)$. In state $(0, 0)$ there are no jobs in the system and a completion can only happen after a job arrives first. Therefore, in this state clock y is not set, but only after a job arrival (in state $(1, 1)$). □

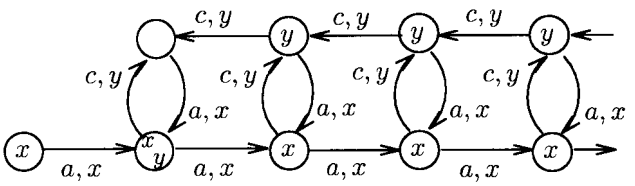

Figure 1: Stoch. automaton of a $G/G/1/\infty$-system

Generalised semi-Markov processes. In Section 2 we have seen a flavour of GSMPs. Actually we will consider a (large) subclass of GSMPs. The main restriction is that the next state is uniquely determined by the present state and the triggered event. In general GSMPs the next state is chosen probabilistically from a set of possible next states. In addition, we assign to each clock a fixed distribution function whereas in general GSMPs such distribution may depend on the history of the system. This is not a severe restriction since one can model each history of a general GSMP by a sequence of sufficiently many events such that each such event marks a relevant point in history. The class of GSMPs with history-independent distribution functions is known as *time-homogeneous* [8]. Finally, sometimes clocks are allowed to have different rates whereas in our case all clocks proceed with the same speed. Different rates are not very usual in simulation, and moreover, under certain conditions, such "multi-rated" GSMPs can be represented by GSMPs where all clock rates equal 1.

The notion of GSMPs that we consider is defined as follows, where it is assumed that initial state z_0 has a single active event e_0. $\mathcal{C}$ is a set of clocks. The dynamics of a GSMP in the following sense is as described in Section 2.

Definition 3. $(Z, z_0, \mathbf{E}, e_0, E, C, N, F)$ is a GSMP with Z, a non-empty set of *states* with $z_0 \in Z$, $\mathbf{E}$, a non-empty set of *events* with $e_0 \in \mathbf{E}$, $E : Z \to \wp_{\text{fin}}(\mathbf{E})$, the *event-assignment* function with $E(z) \neq \varnothing$ for all $z \in Z$ and $E(z_0) = \{ e_0 \}$, $C : \mathbf{E} \to \mathcal{C}$, the *clock-assignment* function, $N : Z \times \mathbf{E} \to Z$, the *next-state* function, and $F : \mathcal{C} \to (\mathbb{R} \to [0, 1])$, the *distribution assignment* function, such that $F(x)(0) = 0$.

GSMPs versus stochastic automata. The relation between stochastic automata and GSMPs (in the above sense) is shown by defining a mapping from GSMPs onto stochastic automata. The existence of this mapping indicates that GSMPs are properly included in stochastic automata. We have proven the correctness of the mapping in the sense that the underlying probabilistic transition systems (that are based on Borel spaces) of a GSMP and its associated stochastic automaton are probabilistically bisimilar [6].

The basic idea of the mapping is to introduce a location as a pair (z, E) where z is a state of the GSMP and E is the set of events that are already active. The initial location is $(N(z_0, e_0), \varnothing)$. For each active event in state z, there is an outgoing edge from any location (z, E). This edge is labelled with event e (i.e. the action) and the set of clocks $\{C(e)\}$. So, events are considered as actions and active events of z are

$$E(z) = \bigcup \{ e \mid (z, E) \xrightarrow{e, \{C(e)\}} \}.$$

Definition 4. The associated stochastic automaton of GSMP $\mathcal{G} = (Z, z_0, \mathbf{E}, e_0, E, C, N, F)$ is defined by $\mathcal{S} = Z \times \wp_{\text{fin}}(\mathbf{E})$ with $s_0 = (N(z_0, e_0), \varnothing)$, $\mathbf{A} = \mathbf{E}$, $\mathcal{C} = \{ C(e) \mid e \in \mathbf{E} \}$, $\kappa(z, E) =$

$\{ C(e) \mid e \in E(z) - E \}$, and F is the same as for $\mathcal{G}$. $\longrightarrow$ is defined by the rule

$$\frac{e \in E(z)}{(z, E) \xrightarrow{e,\{C(e)\}} (N(z,e), E(z) - \{e\})}$$

Due to the fact that $E(z) \neq \varnothing$ for any z, the condition $e \in E(z)$ is always satisfied. There are many locations $(z, E) \in \mathcal{S}$ that are unreachable via $\longrightarrow$. All reachable locations have the form $(N(z,e), E(z) - \{e\})$ for every (reachable) $z \in Z$ and $e \in E(z)$. Remark that for $z' = N(z,e)$ we have $\kappa(z', E(z) - \{e\}) = \{ C(e') \mid e' \in E(z') - (E(z) - \{e\}) \}$, the set of clocks for all newly active events in z'.

As argued before, stochastic automata are more expressive than GSMPs, since stochastic automata do allow non-determinism (two outgoing edges that are enabled at the same time), whereas GSMPs do not. In addition, in the stochastic automaton model clocks may be initialised by arbitrary distributions—including discrete distribution functions—without any restriction. In GSMPs it is required that in any set of active events there is at most one clock x such that $F_x(t)$ is not continuous as a function of t [8].

4 The stochastic process algebra $\mathcal{Q}$

Syntax. Let $\mathbf{A}$ be a set of *actions*, $\mathbf{V}$ a set of *process variables*, and $\mathcal{C}$ a set of clocks with $(x, G) \in \mathcal{C}$ for x a clock name and G an arbitrary probability distribution function. We abbreviate (x, G) by x_G.

Definition 5. The syntax of $\mathcal{Q}$ is defined by:

$$p ::= \mathbf{0} \mid a;p \mid C \mapsto p \mid p + p \mid \{\!|C|\!\}p \mid p \,||_A\, p \mid p[f] \mid X.$$

where $C \subseteq \mathcal{C}$ is finite, $a \in \mathbf{A}$, $A \subseteq \mathbf{A}$, $f : \mathbf{A} \to \mathbf{A}$, and $X \in \mathbf{V}$. A *recursive specification* E is a set of recursive equations of the form $X = p$ for each $X \in \mathbf{V}$, where $p \in \mathcal{Q}$.

Besides the operations used in Section 2 the language incorporates the basic process $\mathbf{0}$, the process that cannot perform any action, and the *renaming* operation $p[f]$, a process that behaves like p except that actions are renamed by function f. A few words on $p + q$ are in order. $p + q$ behaves either as p or q, but not both. At execution the fastest process, i.e. the process that is enabled first, is selected. This is known as the race condition. If this fastest process is not uniquely determined, a non-deterministic selection among the fastest processes is made.

Semantics. To associate a stochastic automaton $SA(p)$ to a given term p in the language, we define the different components of $SA(p)$[1]. In order to define the automaton associated to a parallel composition, we introduce the additional operation $\overline{\mathsf{ck}}$. $\overline{\mathsf{ck}}(p)$ is a process that behaves like p except that no clock is set at the very beginning. As usual in structured operational semantics, a location corresponds to a term. Thus, the set of locations equals $\mathcal{Q} \cup \{\overline{\mathsf{ck}}\}$. The clock setting function κ is defined by induction on the structure of expression: $\kappa(\mathbf{0}) = \kappa(a;p) = \kappa(\overline{\mathsf{ck}}(p)) = \varnothing$, $\kappa(C \mapsto p) = \kappa(p[f]) = \kappa(p)$, $\kappa(p + q) = \kappa(p \,||_A\, q) = \kappa(p) \cup \kappa(q)$, $\kappa(\{\!|C|\!\}p) = C \cup \kappa(p)$ and $\kappa(X) = \kappa(p)$ for $X = p$. The set of edges $\longrightarrow$ between locations is defined as the smallest relation satisfying the rules in Table 1. The function F is defined by $F(x_G) = G$ for each clock x in p. The other components are defined as for the syntax of $\mathcal{Q}$.

Example 6. Using this recipe it can be shown that the semantics of the *System* specification of Section 2 boils down to the (at first sight somewhat complicated) stochastic automaton depicted in Figure 2. Here, empty sets are omitted; in particular b stands for $b, \varnothing$. Although the state space of this automaton is somewhat larger than that of the direct representation in Figure 1, this does not have a serious impact on the efficiency of stochastic simulation. Since in our semantics a state corresponds to a term, simulation can be carried out on the basis of expressions rather than

[1] Here we assume that p does not contain any name clashes of clock variables. This not a severe restriction since terms that suffer from such name clash can always be properly renamed into a term without such name clash [6].

Table 1: Stochastic automata for $\mathcal{O}$ $(X = p \in E)$

$$a; p \xrightarrow{a,\emptyset} p \qquad \frac{p \xrightarrow{a,C} p'}{p + q \xrightarrow{a,C} p' \quad q + p \xrightarrow{a,C} p'}$$

$$\frac{p \xrightarrow{a,C'} p'}{\{\!|C|\!\}p \xrightarrow{a,C'} p'} \qquad \frac{p \xrightarrow{a,C} p'}{p[f] \xrightarrow{f(a),C} p'[f]}$$

$$\frac{p \xrightarrow{a,C'} p'}{C \mapsto p \xrightarrow{a,C \cup C'} p'} \qquad \frac{p \xrightarrow{a,C} p'}{\overline{\mathsf{ck}}(p) \xrightarrow{a,C} p'}$$

$$\frac{p \xrightarrow{a,C} p'}{X \xrightarrow{a,C} p'} \qquad \frac{p \xrightarrow{a,C} p'}{p \,||_A\, q \xrightarrow{a,C} p' \,||_A\, \overline{\mathsf{ck}}(q) \quad q \,||_A\, p \xrightarrow{a,C} \overline{\mathsf{ck}}(q) \,||_A\, p'} \; (a \notin A)$$

$$\frac{p \xrightarrow{a,C} p' \quad q \xrightarrow{a,C'} q'}{p \,||_A\, q \xrightarrow{a,C \cup C'} p' \,||_A\, q'} \; (a \in A)$$

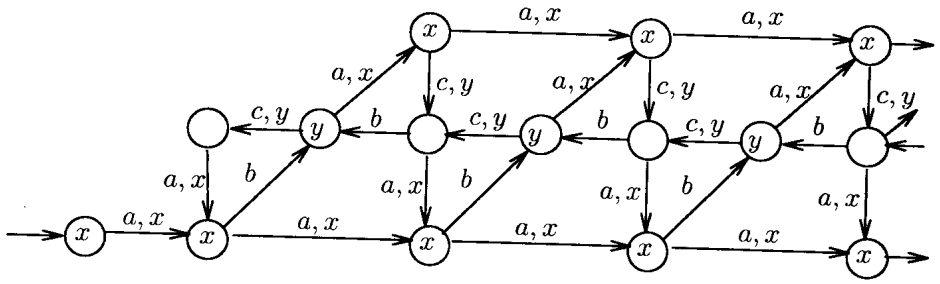

Figure 2: Stochastic automaton of the compositional $G/G/1/\infty$ specification

using their semantic representations. This allows on-the-fly simulation, that is, simulation while constructing the state space. In this approach the unreachable locations will not be visited. For instance, for locations in which both an immediate (i.e. an action equipped with no clocks) and a non-immediate action are enabled, the non-immediate transition will never be traversed. The corresponding reduction of the state space can also be obtained by syntactical transformation as shown in Example 9. □

It turns out that stochastic automata and the language $\mathcal{O}$ are equally expressive [6]. This means that for any (finitely branching) stochastic automaton a corresponding (guarded recursive) term in the language can be given whose reachable part of its stochastic automaton is identical to the stochastic automaton at hand, up to renaming of clocks. A recursive specification E is *guarded* if $X = p \in E$ implies that all variables in p appear in a context of a prefix. A stochastic automaton is finitely branching if for every location the set of outgoing edges is finite.

Structural bisimulation. For process algebras many equivalences and pre-orders have been defined to compare specifications. One of the most interesting equivalence relations is bisimulation [15]. The following notion of bisimulation decides the equivalence of stochastic automata on the basis of their structure. Weaker notions of bisimulation are defined in [6].

Definition 7. Let $(\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \longrightarrow, \kappa, F)$ be a stochastic automaton. $R \subseteq \mathcal{S} \times \mathcal{S}$ is a *structural*

bisimulation if, R is symmetric and whenever $s_1 R s_2$, for all $a \in \mathbf{A}$, $C \subseteq \mathcal{C}$, we have:

1. $s_1 \xrightarrow{a,C} s_1'$ implies $\exists s_2'.\ s_2 \xrightarrow{a,C} s_2'$ and $s_1' R s_2'$;
2. $\kappa(s_1) = \kappa(s_2)$

If R is a structural bisimulation such that $s_1 R s_2$, we write $s_1 \leftrightarrows\!\!\sim s_2$ and call s_1 and s_2 structurally bisimilar.

Two stochastic automata SA_1 and SA_2 are *structurally bisimilar*, notation $SA_1 \leftrightarrows\!\!\sim SA_2$, if their respective initial locations are structurally bisimilar on the disjoint union of SA_1 and SA_2. If we omit the clock-related information, we obtain the usual (strong) bisimulation relation on transition systems [15]. Terms p and q are structurally bisimilar if and only if $SA(p) \leftrightarrows\!\!\sim SA(q)$. The relation $\leftrightarrows\!\!\sim$ is a congruence for $\hat{\Diamond}$ [6]. This means that for any term in our language a sub-term p may be replaced by its bisimilar equivalent q such that a bisimilar term results.

Equational reasoning. Rather than proving $p \leftrightarrows\!\!\sim q$ using their semantical interpretation it is often more convenient to use rules defined on the syntax of p and q that are known to preserve (in our case) $\leftrightarrows\!\!\sim$. This enables the transformation and comparison of terms at a purely syntactical level. Some typical axioms are

$$\begin{aligned} (p+q)+r &= p+(q+r) \\ C \mapsto (C' \mapsto p) &= C \cup C' \mapsto p \\ C \mapsto \{\!|C'|\!\}p &= \{\!|C'|\!\}C \mapsto p \text{ if } C \cap C' = \varnothing. \end{aligned}$$

In [6] a complete and sound axiomatisation of structural bisimulation for $\hat{\Diamond}$ is presented for finite terms. Using these axioms any term p can be converted into a canonical form which has the shape $\{\!|C|\!\}(\sum C_i \mapsto a_i; p_i)$ where p_i are terms in canonical form and $\sum$ is the usual generalisation of choice: $\sum_{0<i\leqslant n} p_i$ equals $p_1 + \ldots + p_n$ for $n{>}0$, and $\mathbf{0}$ for $n{=}0$.

An essential law in traditional process algebras is the expansion law. This law allows one to reduce parallel composition in terms of prefix and choice, and has proven to be of crucial importance for verification purposes. A stochastic equivalent of this law can be derived for our language. In fact, the expansion law is inherent in our model and follows from the way in which parallel composition is defined. It can be derived using the axioms in [6].

Theorem 8. (Expansion Law) Let $p, q \in \hat{\Diamond}$ such that $p = \{\!|C|\!\}p'$ and $q = \{\!|C'|\!\}q'$ with $p' = \sum C_i \mapsto a_i; p_i$ and $q' = \sum C_j' \mapsto b_j; q_j$. Suppose $p \,||_A\, q$ does not contain name clashes. Then $p \,||_A\, q$ equals

$$\begin{aligned} \{\!|C \cup C'|\!\}\Big(& \textstyle\sum_{a_i \notin A} C_i \mapsto a_i; (p_i \,||_A\, q') \\ + & \textstyle\sum_{b_j \notin A} C_j' \mapsto b_j; (p' \,||_A\, q_j) \\ + & \textstyle\sum_{a_i = b_j \in A} (C_i \cup C_j') \mapsto a_i; (p_i \,||_A\, q_j) \Big). \end{aligned}$$

Example 9. Using structural and probabilistic bisimulation [6] we are able to formally relate Figure 2 to Figure 1 in the following way. (The following transformations could also be carried out using equational laws, but this is omitted due to space reasons.) If in a location both an immediate action and a non-immediate action are possible, then the latter will never be taken since it has to be delayed first. This allows one to remove the locations in the lower row of Figure 2, except for the two leftmost locations. The thus obtained automaton is depicted in Figure 3. The only difference with Figure 1 are the b-transitions that are used for the sole purpose of synchronising the *Queue* and *Server* processes. If, as a last step, we would copy the leftmost location that contains clock y and subsequently aggregate locations appropriately such that the b-transitions only occur inside aggregates, then we obtain the automaton of Figure 1. This latter transformation can be formalised using the common notion of abstraction in process algebra and weak bisimulation equivalence [15], a notion of equivalence that allows one to abstract from internal moves. This approach is applied to Markovian queues in [10], and is for our setting an interesting subject for further work. □

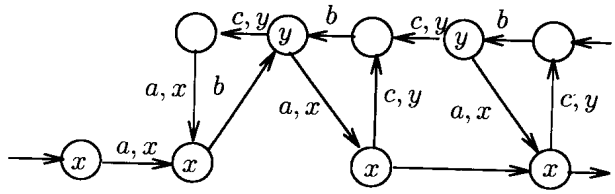

Figure 3: Reduced automaton

5 Related work

Other stochastic process algebras. Since 1990 extensions of process algebras have been investigated in which the time of actions is determined by (continuous) distribution functions. In languages like TIPP [11], PEPA [12] and EMPA [2] the time of actions is controlled by exponential distributions. The elementary operator in these languages is $a_\lambda; p$ with rate λ. This corresponds to $\{\!|x_F|\!\}\{x_F\} \mapsto a; p$ with $F(t) = 1 - e^{-\lambda t}$. Due to the memoryless property of exponential distributions the semantics of these languages can adequately be described using labelled transition systems that closely resemble continuous-time Markov chains. Various useful links between process algebras and Markov chains have thus been established, e.g. between bisimulation and lumping [12].

In fact, our presented approach can be considered as a generalisation of this line of research using GSMPs rather than Markov chains. To our knowledge the use of a process algebra to specify GSMPs is novel. [14] used GSMPs indirectly: they map an extended process algebra onto event structures, and obtain for a subclass of event structures a GSMP. For recursive processes infinite event structures are obtained which makes this approach less suited for the use of efficient regenerative simulation techniques. The finite representations we obtain do not suffer from this problem.

Process algebra and discrete-event simulation. Harrison & Strulo [9] developed a stochastic process algebra to formally describe discrete-event simulation. Typically their semantic objects are highly infinite. Although their work is somehow related to ours, stochastic automata appear to be more intuitive and resemble more closely the conceptual ideas of simulation languages. In particular, measure theory only plays a role in our case when defining the formal interpretation of stochastic automata.

Pooley [16] investigates the mapping of a high-level language for describing discrete-event simulation models, baptised extended activity diagrams, onto the timed process algebra TCCS and the simulation language Demos [3]. Using this framework Pooley is able to check certain properties of a model a priori to simulation, by analysing the (T)CCS specification. In this work distribution functions are neglected and the use of process algebra is quite different from ours.

Birtwistle and Tofts use process algebras, basically CCS and its synchronous variant SCCS, to provide a denotational semantics of Demos [4]. They focus on analysing properties like absence of deadlock and livelock and do not consider any timing aspects.

6 Concluding remarks

In this paper we presented a novel process algebra suitable for specifying GSMPs in a compositional way. The concept of stochastic automata has been introduced and is shown to properly contain a large class of GSMPs. Since our process algebra ♤ and stochastic automata are equally expressive, this class of GSMPs is also a subset of ♤. Using equational laws—like the presented expansion law that allows to reduce parallel composition—GSMP specifications can be simplified and compared syntactically.

We have currently implemented a prototypical tool that allows us to simulate specifications written in ♤. The simulation algorithm takes a ♤ specification and an additional process to resolve

possible non-determinism in this process as input, and automatically generates simulation runs. We applied our prototype to model and analyse a multi-processor system that is vulnerable to failures [7].

References

[1] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[2] M. Bernardo and R. Gorrieri. A Tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, **202**(1-2):1–54, 1998.

[3] G.M. Birtwistle. *Discrete Event Modelling on Simula.* MacMillan, 1979.

[4] G.M. Birtwistle and C. Tofts. Process semantics for simulation. Technical report, University of Swansea, 1996.

[5] P.R. D'Argenio and E. Brinksma. A calculus for timed automata (Extended abstract). In *Proceedings FTRTFT'96*, LNCS 1135, pp 110–129. Springer-Verlag, 1996.

[6] P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. An algebraic approach to the specification of stochastic systems (Extended abstract). In *Proceedings PROCOMET'98.* Chapman & Hall, 1998.

[7] P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. Specification and Analysis of Soft Real-Time Systems: Quantity and Quality. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, Phoenix, Arizona, 104–114, IEEE Society Press, 1999.

[8] P.W. Glynn. A GSMP formalism for discrete event simulation. *Proceedings of the IEEE*, 77(1):14–23, 1989.

[9] P.G. Harrison and B. Strulo. Stochastic process algebra for discrete event simulation. In *Quantitative Methods in Parallel Systems*, pp 18–37. Springer, 1995.

[10] H. Hermanns, M. Rettelbach and T. Weiss. Formal characterisation of immediate actions in SPA with nondeterministic branching. *The Computer Journal*, 38(7):530–542, 1995.

[11] H. Hermanns, U. Herzog and V. Mertsiotakis. Stochastic process algebras – Between LOTOS and Markov chains. *Computer Networks & ISDN Systems*, 1998.

[12] J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.

[13] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[14] J.-P. Katoen, E. Brinksma, D. Latella, and R. Langerak. Stochastic simulation of event structures. In *Proceedings PAPM'96*, pp 21–40. CLUT Press, 1996.

[15] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[16] R.J. Pooley. Integrating behavioural and simulation modelling. In *Quantitative Evaluation of Computing and Communication Systems*, LNCS 977, pp 102–116. Springer-Verlag, 1995.

[17] M.Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proceedings* 26^{th} *FOCS*, pp 327–338. IEEE Comp. Soc. Press, 1985.