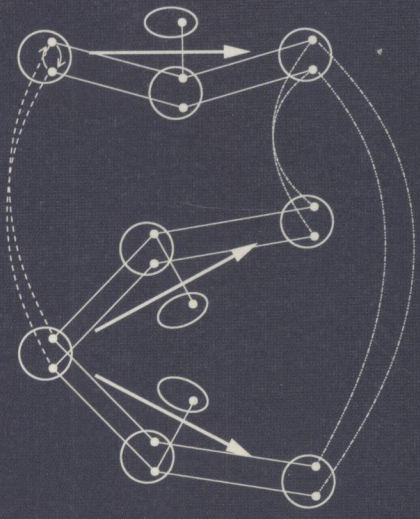
Tutorial

LNCS 3465

Marco Bernardo
Alessandro Bogliolo (Eds.)

Formal Methods for Mobile Computing

5th International School on Formal Methods for the Design of Computer, Communication, and Software Systems SFM-Moby 2005, Bertinoro, Italy, April 2005 Advanced Lectures



Springer

Marco Bernardo Alessandro Bogliolo (Eds.)

Formal Methods for Mobile Computing

5th International School on Formal Methods for the Design of Computer, Communication, and Software Systems
SFM-Moby 2005
Bertinoro, Italy, April 26-30, 2005
Advanced Lectures

Springer

Volume Editors

Marco Bernardo
Alessandro Bogliolo
Università degli Studi di Urbino "Carlo Bo"
Istituto di Scienze e Tecnologie dell'Informazione
Piazza della Repubblica 13, 61029 Urbino, Italy
E-mail: {bernardo, bogliolo}@sti.uniurb.it

Library of Congress Control Number: 2005924063

CR Subject Classification (1998): D.2, D.3, F.3, C.3, C.2.4

ISSN 0302-9743
ISBN-10 3-540-25697-0 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-25697-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11419822 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3465

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Preface

This volume collects a set of papers accompanying the lectures of the fifth edition of the International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM).

This series of schools addresses the use of formal methods in computer science as a prominent approach to the rigorous design of computer, communication and software systems. The main aim of the SFM series is to offer a good spectrum of current research in foundations as well as applications of formal methods, which can be of help for graduate students and young researchers who intend to approach the field.

SFM 2005 (Moby) was devoted to formal methods and tools for the design of mobile systems and mobile communication infrastructures. This volume is organized into four parts related to mobile computing, which cover models and languages, scalability and performance, dynamic power management, and middleware support. Each part is composed of two papers.

The opening paper by Montanari and Pistore gives an overview of history-dependent automata, an extension of ordinary automata that overcomes their limitations in dealing with named calculi. In particular, the authors show that history-dependent automata allow for a compact representation of π-calculus processes, which is suitable both for theoretical investigations and for the verification of models of agents and code mobility. Bettini and De Nicola's paper presents X-KLAIM, an experimental programming language specifically designed to develop distributed systems composed of several components interacting through multiple distributed tuple spaces and mobile code. Through a series of examples, the authors show that many mobile code programming paradigms can be naturally implemented by means of the considered language, which combines explicit localities as first-class data with coordination primitives.

Gerla, Chen, Lee, Zhou, Chen, Yang and Das provide an introduction to MANET, a mobile ad hoc wireless network established for a special, often extemporaneous service customized to applications. After emphasizing the self-configurability, mobility and scalability attributes of MANET, the authors concentrate on mobility and show its impact on protocols and operations. Grassi presents an overview of the performance issues raised by the high variability and heterogeneity of mobile systems, together with some approaches to the careful planning of the performance validation of such systems. The author then focuses on the definition of model-based transformations from design-oriented models to analysis-oriented models that comprise non-functional attributes.

Acquaviva, Aldini, Bernardo, Bogliolo, Bontà and Lattanzi illustrate in their paper a methodology for predicting the impact on the overall system functionality and efficiency of the introduction of a dynamic power management policy within a battery-powered mobile device. The predictive methodology relies on a com-

bination of formal description techniques, noninterference analysis, and performance evaluation to properly tune the dynamic power manager operation rates. The methodology is then used by Acquaviva, Bontà and Lattanzi in the framework of the IEEE 802.11 standard, in order to provide a power-accurate model of a wireless network interface card that allows the energy/performance trade-off to be studied as a function of traffic patterns imposed by the applications.

Lattanzi, Acquaviva and Bogliolo address the limited storage memory of wireless mobile terminals through the concept of network virtual memory. The authors first compare the performance and energy of network swapping with those of local swapping on microdrives and flash memories, then present an infrastructure providing efficient remote memory access to mobile terminals. The closing paper, by Corradini and Merelli, reports on Hermes, a middleware system for the design and the execution of activity-based applications in distributed environments. While middleware for mobile computing has typically been developed to support physical and logical mobility, Hermes provides an integrated environment where application-domain experts can focus on designing the activity workflow.

We believe that this book offers a quite comprehensive view of what has been done and what is going on worldwide at present in the field of formal methods for mobile computing. We wish to thank all the lecturers and all the participants for a lively and fruitful school. We also wish to thank the whole staff of the University Residential Center of Bertinoro (Italy) for the organizational and administrative support, as well as the Regione Marche, which sponsored the school within the CIPE 36/2002 framework.

April 2005

Marco Bernardo and Alessandro Bogliolo
SFM 2005 (Moby) Directors

Lecture Notes in Computer Science

For information about Vols. 1–3358

please contact your bookseller or Springer

Vol. 3467: J. Giesl (Ed.), Term Rewriting and Applications. XIII, 517 pages. 2005.

Vol. 3465: M. Bernardo, A. Bogliolo (Eds.), Formal Methods for Mobile Computing. VII, 271 pages. 2005.

Vol. 3461: P. Urzyczyn (Ed.), Typed Lambda Calculi and Applications. XI, 433 pages. 2005.

Vol. 3459: R. Kimmel, N. Sochen, J. Weickert (Eds.), Scale Space and PDE Methods in Computer Vision. XI, 634 pages. 2005.

Vol. 3456: H. Rust, Operational Semantics for Timed Systems. XII, 223 pages. 2005.

Vol. 3455: H. Treharne, S. King, M. Henson, S. Schneider (Eds.), ZB 2005: Formal Specification and Development in Z and B. XV, 493 pages. 2005.

Vol. 3454: J.-M. Jacquet, G.P. Picco (Eds.), Coordination Models and Languages. X, 299 pages. 2005.

Vol. 3453: L. Zhou, B.C. Ooi, X. Meng (Eds.), Database Systems for Advanced Applications. XXVII, 929 pages. 2005.

Vol. 3452: F. Baader, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning. XI, 562 pages. 2005. (Subseries LNAI).

Vol. 3450: D. Hutter, M. Ullmann (Eds.), Security in Pervasive Computing. XI, 239 pages. 2005.

Vol. 3449: F. Rothlauf, J. Branke, S. Cagnoni, D.W. Corne, R. Drechsler, Y. Jin, P. Machado, E. Marchiori, J. Romero, G.D. Smith, G. Squillero (Eds.), Applications on Evolutionary Computing. XX, 631 pages. 2005.

Vol. 3448: G.R. Raidl, J. Gottlieb (Eds.), Evolutionary Computation in Combinatorial Optimization. XI, 271 pages. 2005.

Vol. 3447: M. Keijzer, A. Tettamanzi, P. Collet, J.v. Hemert, M. Tomassini (Eds.), Genetic Programming. XIII, 382 pages. 2005.

Vol. 3444: M. Sagiv (Ed.), Programming Languages and Systems. XIII, 439 pages. 2005.

Vol. 3443: R. Bodik (Ed.), Compiler Construction. XI, 305 pages. 2005.

Vol. 3442: M. Cerioli (Ed.), Fundamental Approaches to Software Engineering. XIII, 373 pages. 2005.

Vol. 3441: V. Sassone (Ed.), Foundations of Software Science and Computational Structures. XVIII, 521 pages. 2005.

Vol. 3440: N. Halbwachs, L.D. Zuck (Eds.), Tools and Algorithms for the Construction and Analysis of Systems. XVII, 588 pages. 2005.

Vol. 3439: R.H. Deng, F. Bao, H. Pang, J. Zhou (Eds.), Information Security Practice and Experience. XII, 424 pages. 2005.

Vol. 3437: T. Gschwind, C. Mascolo (Eds.), Software Engineering and Middleware. X, 245 pages. 2005.

Vol. 3436: B. Bouyssounouse, J. Sifakis (Eds.), Embedded Systems Design. XV, 492 pages. 2005.

Vol. 3434: L. Brun, M. Vento (Eds.), Graph-Based Representations in Pattern Recognition. XII, 384 pages. 2005.

Vol. 3433: S. Bhalla (Ed.), Databases in Networked Information Systems. VII, 319 pages. 2005.

Vol. 3432: M. Beigl, P. Lukowicz (Eds.), Systems Aspects in Organic and Pervasive Computing - ARCS 2005. X, 265 pages. 2005.

Vol. 3431: C. Dovrolis (Ed.), Passive and Active Network Measurement. XII, 374 pages. 2005.

Vol. 3429: E. Andres, G. Damiand, P. Lienhardt (Eds.), Discrete Geometry for Computer Imagery. X, 428 pages. 2005.

Vol. 3427: G. Kotsis, O. Spaniol, Wireless Systems and Mobility in Next Generation Internet. VIII, 249 pages. 2005.

Vol. 3423: J.L. Fiadeiro, P.D. Mosses, F. Orejas (Eds.), Recent Trends in Algebraic Development Techniques. VIII, 271 pages. 2005.

Vol. 3422: R.T. Mittermeir (Ed.), From Computer Literacy to Informatics Fundamentals. X, 203 pages. 2005.

Vol. 3421: P. Lorenz, P. Dini (Eds.), Networking - ICN 2005, Part II. XXXV, 1153 pages. 2005.

Vol. 3420: P. Lorenz, P. Dini (Eds.), Networking - ICN 2005, Part I. XXXV, 933 pages. 2005.

Vol. 3419: B. Faltings, A. Petcu, F. Fages, F. Rossi (Eds.), Constraint Satisfaction and Constraint Logic Programming. X, 217 pages. 2005. (Subseries LNAI).

Vol. 3418: U. Brandes, T. Erlebach (Eds.), Network Analysis. XII, 471 pages. 2005.

Vol. 3416: M. Böhlen, J. Gamper, W. Polasek, M.A. Wimmer (Eds.), E-Government: Towards Electronic Democracy. XIII, 311 pages. 2005. (Subseries LNAI).

Vol. 3415: P. Davidsson, B. Logan, K. Takadama (Eds.), Multi-Agent and Multi-Agent-Based Simulation. X, 265 pages. 2005. (Subseries LNAI).

Vol. 3414: M. Morari, L. Thiele (Eds.), Hybrid Systems: Computation and Control. XII, 684 pages. 2005.

Vol. 3412: X. Franch, D. Port (Eds.), COTS-Based Software Systems. XVI, 312 pages. 2005.

Vol. 3411: S.H. Myaeng, M. Zhou, K.-F. Wong, H.-J. Zhang (Eds.), Information Retrieval Technology. XIII, 337 pages. 2005.

Vol. 3410: C.A. Coello Coello, A. Hernández Aguirre, E. Zitzler (Eds.), Evolutionary Multi-Criterion Optimization. XVI, 912 pages. 2005.

Vol. 3409: N. Guelfi, G. Reggio, A. Romanovsky (Eds.), Scientific Engineering of Distributed Java Applications. X, 127 pages. 2005.

Vol. 3408: D.E. Losada, J.M. Fernández-Luna (Eds.), Advances in Information Retrieval. XVII, 572 pages. 2005.

Vol. 3407: Z. Liu, K. Araki (Eds.), Theoretical Aspects of Computing - ICTAC 2004. XIV, 562 pages. 2005.

Vol. 3406: A. Gelbukh (Ed.), Computational Linguistics and Intelligent Text Processing. XVII, 829 pages. 2005.

Vol. 3404: V. Diekert, B. Durand (Eds.), STACS 2005. XVI, 706 pages. 2005.

Vol. 3403: B. Ganter, R. Godin (Eds.), Formal Concept Analysis. XI, 419 pages. 2005. (Subseries LNAI).

Vol. 3401: Z. Li, L.G. Vulkov, J. Waśniewski (Eds.), Numerical Analysis and Its Applications. XIII, 630 pages. 2005.

Vol. 3399: Y. Zhang, K. Tanaka, J.X. Yu, S. Wang, M. Li (Eds.), Web Technologies Research and Development - APWeb 2005. XXII, 1082 pages. 2005.

Vol. 3398: D.-K. Baik (Ed.), Systems Modeling and Simulation: Theory and Applications. XIV, 733 pages. 2005. (Subseries LNAI).

Vol. 3397: T.G. Kim (Ed.), Artificial Intelligence and Simulation. XV, 711 pages. 2005. (Subseries LNAI).

Vol. 3396: R.M. van Eijk, M.-P. Huget, F. Dignum (Eds.), Agent Communication. X, 261 pages. 2005. (Subseries LNAI).

Vol. 3395: J. Grabowski, B. Nielsen (Eds.), Formal Approaches to Software Testing. X, 225 pages. 2005.

Vol. 3394: D. Kudenko, D. Kazakov, E. Alonso (Eds.), Adaptive Agents and Multi-Agent Systems III. VIII, 313 pages. 2005. (Subseries LNAI).

Vol. 3393: H.-J. Kreowski, U. Montanari, F. Orejas, G. Rozenberg, G. Taentzer (Eds.), Formal Methods in Software and Systems Modeling. XXVII, 413 pages. 2005.

Vol. 3392: D. Seipel, M. Hanus, U. Geske, O. Bartenstein (Eds.), Applications of Declarative Programming and Knowledge Management. X, 309 pages. 2005. (Subseries LNAI).

Vol. 3391: C. Kim (Ed.), Information Networking. XVII, 936 pages. 2005.

Vol. 3390: R. Choren, A. Garcia, C. Lucena, A. Romanovsky (Eds.), Software Engineering for Multi-Agent Systems III. XII, 291 pages. 2005.

Vol. 3389: P. Van Roy (Ed.), Multiparadigm Programming in Mozart/OZ. XV, 329 pages. 2005.

Vol. 3388: J. Lagergren (Ed.), Comparative Genomics. VII, 133 pages. 2005. (Subseries LNBI).

Vol. 3387: J. Cardoso, A. Sheth (Eds.), Semantic Web Services and Web Process Composition. VIII, 147 pages. 2005.

Vol. 3386: S. Vaudenay (Ed.), Public Key Cryptography - PKC 2005. IX, 436 pages. 2005.

Vol. 3385: R. Cousot (Ed.), Verification, Model Checking, and Abstract Interpretation. XII, 483 pages. 2005.

Vol. 3383: J. Pach (Ed.), Graph Drawing. XII, 536 pages. 2005.

Vol. 3382: J. Odell, P. Giorgini, J.P. Müller (Eds.), Agent-Oriented Software Engineering V. X, 239 pages. 2005.

Vol. 3381: P. Vojtáš, M. Bieliková, B. Charron-Bost, O. Sýkora (Eds.), SOFSEM 2005: Theory and Practice of Computer Science. XV, 448 pages. 2005.

Vol. 3380: C. Priami, Transactions on Computational Systems Biology I. IX, 111 pages. 2005. (Subseries LNBI).

Vol. 3379: M. Hemmje, C. Niederee, T. Risse (Eds.), From Integrated Publication and Information Systems to Information and Knowledge Environments. XXIV, 321 pages. 2005.

Vol. 3378: J. Kilian (Ed.), Theory of Cryptography. XII, 621 pages. 2005.

Vol. 3377: B. Goethals, A. Siebes (Eds.), Knowledge Discovery in Inductive Databases. VII, 190 pages. 2005.

Vol. 3376: A. Menezes (Ed.), Topics in Cryptology – CT-RSA 2005. X, 385 pages. 2005.

Vol. 3375: M.A. Marsan, G. Bianchi, M. Listanti, M. Meo (Eds.), Quality of Service in Multiservice IP Networks. XIII, 656 pages. 2005.

Vol. 3374: D. Weyns, H.V.D. Parunak, F. Michel (Eds.), Environments for Multi-Agent Systems. X, 279 pages. 2005. (Subseries LNAI).

Vol. 3372: C. Bussler, V. Tannen, I. Fundulaki (Eds.), Semantic Web and Databases. X, 227 pages. 2005.

Vol. 3371: M.W. Barley, N. Kasabov (Eds.), Intelligent Agents and Multi-Agent Systems. X, 329 pages. 2005. (Subseries LNAI).

Vol. 3370: A. Konagaya, K. Satou (Eds.), Grid Computing in Life Science. X, 188 pages. 2005. (Subseries LNBI).

Vol. 3369: V.R. Benjamins, P. Casanovas, J. Breuker, A. Gangemi (Eds.), Law and the Semantic Web. XII, 249 pages. 2005. (Subseries LNAI).

Vol. 3368: L. Paletta, J.K. Tsotsos, E. Rome, G.W. Humphreys (Eds.), Attention and Performance in Computational Vision. VIII, 231 pages. 2005.

Vol. 3367: W.S. Ng, B.C. Ooi, A. Ouksel, C. Sartori (Eds.), Databases, Information Systems, and Peer-to-Peer Computing. X, 231 pages. 2005.

Vol. 3366: I. Rahwan, P. Moraitis, C. Reed (Eds.), Argumentation in Multi-Agent Systems. XII, 263 pages. 2005. (Subseries LNAI).

Vol. 3365: G. Mauri, G. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing. IX, 415 pages. 2005.

Vol. 3363: T. Eiter, L. Libkin (Eds.), Database Theory - ICDT 2005. XI, 413 pages. 2004.

Vol. 3362: G. Barthe, L. Burdy, M. Huisman, J.-L. Lanet, T. Muntean (Eds.), Construction and Analysis of Safe, Secure, and Interoperable Smart Devices. IX, 257 pages. 2005.

Vol. 3361: S. Bengio, H. Bourlard (Eds.), Machine Learning for Multimodal Interaction. XII, 362 pages. 2005.

Vol. 3360: S. Spaccapietra, E. Bertino, S. Jajodia, R. King, D. McLeod, M.E. Orlowska, L. Strous (Eds.), Journal on Data Semantics II. XI, 223 pages. 2005.

Vol. 3359: G. Grieser, Y. Tanaka (Eds.), Intuitive Human Interfaces for Organizing and Accessing Intellectual Assets. XIV, 257 pages. 2005. (Subseries LNAI).

Table of Contents

History-Dependent Automata: An Introduction

Ugo Montanari[1] and Marco Pistore[2]

[1] University of Trento, Italy
marco.pistore@unitn.it
[2] University of Pisa, Italy
ugo@di.unipi.it

Abstract. In this paper we give an overview of History Dependent Automata, an extension of ordinary automata that overcomes their limitations in dealing with named calculi. In a named calculus, the observations labelling the transitions of a system may contain names which represent features such as communication channels, node identifiers, or the locations of the system. An example of named calculus is π-calculus, which has the ability of sending channel names as messages and thus of dynamically reconfiguring process acquaintances and of modeling agents and code mobility. We show that History-Dependent Automata allow for a compact representation of π-calculus processes which is suitable both for theoretical investigations and for practical purposes such as verification.

1 Introduction

In the context of process calculi (e.g., Milner's CCS [Mil89]), *automata* (or *labelled transition systems*) are often used as operational models. They allow for a simple representation of process behavior, and many concepts and theoretical results for these process calculi are independent from the particular syntax of the languages and can be formulated directly on automata. In particular, this is true for the *behavioral equivalences* and preorders which have been defined for these languages, like bisimulation equivalence [Mil89, Par80]: in fact they take into account only the labelled actions a process can perform. Automata are also important from an algorithmic point of view: efficient and practical techniques and tools for verification [IP96, Mad92] have been developed for *finite-state* automata. Finite state verification is successful here, differently than in ordinary programming, since the control part and the data part of protocols and hardware components can be often cleanly separated, and the control part is usually both quite complex and finite state. Particularly interesting is also the possibility to associate to each automaton — and, consequently, to each process — a *minimal realization*, i.e., a minimal automaton which is equivalent to the original one. This is important both from a theoretical point of view — equivalent systems give rise to the same (up to isomorphism) minimal realization — and from a practical point of view — smaller state spaces can be obtained.

This ideal situation, however, does not apply to all process calculi. In the case of *named calculi*, in particular, infinite-state transition systems are generated instead, also

M. Bernardo and A. Bogliolo (Eds.): SFM-Moby 2005, LNCS 3465, pp. 1–28, 2005.

by very simple processes. In a *named calculus*, the observations labelling the transitions of a system may contain names which are used to identify different features of the modeled system, such as the communication channels, the agents participating to the system, or the locations describing the spatial structure of the system. A quite interesting example of named calculus is π-calculus [MPW92, Mil93]. It has the ability of sending channel names as messages and thus of dynamically reconfiguring process acquaintances. More importantly, π-calculus names can model objects (in the sense of object oriented programming [Wal95]) and name sending thus models higher order communication and mobile code [San93b].

The operational semantics of π-calculus is given via a labelled transition system. However labelled transition systems are not fully adequate to deal with the peculiar features of the calculus and complications occur in the creation of new channels. Consider process $p = (\nu y)\,\bar{x}y.y(z).\mathbf{0}$. Channel y is initially a local channel for the process (prefix $(\nu y)\_$ is the operator for scope restriction) and no global communication can occur on it. Action $\bar{x}y$, however, which corresponds to the output of name y on the global channel x, makes name y known also outside the process; after the output has taken place, channel y can be used for further communications, and, in fact, y is used in $y(z).\mathbf{0}$ as the channel for an input transition: so the communication of a restricted name creates a new public channel for the process. The creation of this new channel is represented in the ordinary semantics of the π-calculus by means of an infinite bunch of transitions of the form $p \xrightarrow{\bar{x}(w)} w(z).\mathbf{0}$, where w is any name that is not already in use (i.e., $w \neq x$ in our example, since x is the only name in use by p; notice that $w = y$ is just a particular case). This way to represent the creation of new names has some disadvantages: first of all, also very simple π-calculus processes, like p, give rise to infinite-state and infinite-branching transition systems. Moreover, equivalent processes do not necessarily have the same sets of channel names; so, there are processes q equivalent to p which cannot use y as the name for the newly created channel. Special rules are needed in the definition of bisimulation to take care of this problem and, as a consequence, standard theories and algorithms do not apply to π-calculus.

The ideal situation of ordinary automata can (at least in part) be recovered also in the field of named calculi, by introducing a new operational model which is adequate to deal with these languages, and by extending to this new model (part of) the classical theory for ordinary automata. As model we propose the *history-dependent automata* (*HD-automata* in brief). As ordinary automata, they are composed of states and of transitions between states. To deal with the peculiar problems of named calculi, however, states and transitions are enriched with sets of local names: in particular, each transition can refer to the names associated to its source state but can also generate new names, which can then appear in the destination state. In this manner, the names are not global and static, as in ordinary labelled transition systems, but they are explicitly represented within states and transitions and can be dynamically created.

This explicit representation of names permits an adequate representation of the behavior of named processes. In particular, π-calculus processes can be translated into HD-automata and a first sign of the adequacy of HD-automata for dealing with π-calculus is that a large class of *finitary* π-calculus processes can be represented by finite-state HD-automata. We also give a general definition of bisimulation for HD-automata.

An important result is that this general bisimulation equates the HD-automata obtained from two π-calculus processes if and only if the processes are bisimilar according to the ordinary π-calculus bisimilarity relation. The most interesting result on HD-automata is that they can be minimized. It is possible to associate to each HD-automaton a minimal realization, namely a minimal HD-automaton that is bisimilar to the initial one. As in the case of ordinary automata, this possibility is important from a theoretical but also from a practical point of view.

In this paper we give an introduction to HD-automata. Some of the basic results on ordinary automata and an overview of the π-calculus are briefly presented in Section 2. Section 3 introduces HD-automata, defines bisimulation on HD-automata, and presents the translation of π-calculus processes to HD-automata. Section 4 describes how HD-automata can be minimized by taking into account symmetries on the names enriching states and transitions. Finally, in Section 5 we propose some concluding remarks. Further results on HD-automata (as well as the proofs of the results that we present in this paper) can be found in [MP98b, MP98a, MP99, MP00].

2 Background

2.1 Ordinary Automata

Automata have been defined in a large variety of manners. We choose the following definition since it is very natural and since, as we will see, it can be easily modified to define HD-automata.

Definition 1 (ordinary automata). *An* automaton $\mathcal{A}$ *is defined by:*

- *a set* L *of* labels*;*
- *a set* Q *of* states*;*
- *a set* T *of* transitions*;*
- *two functions* $s, d : T \to Q$ *that associate a* source *and a* destination *state to each transition;*
- *a function* $o : T \to L$ *which associates a label to each transition;*
- *an* initial state $q_0 \in Q$.

Given a transition $t \in T$*, we write* $t : q \xrightarrow{l} q'$ *if* $s(t) = q$, $d(t) = q'$ *and* $o(t) = l$.

Notation 2. *To represent the components of an automaton we will use the name of the automaton as subscript; so, for instance,* $Q_{\mathcal{B}}$ *are the states of automaton* $\mathcal{B}$ *and* $d_{\mathcal{B}}$ *is its destination function. In the case of automaton* $\mathcal{A}_x$*, we will simply write* Q_x *and* d_x *rather than* $Q_{\mathcal{A}_x}$ *and* $d_{\mathcal{A}_x}$*. Moreover, the subscripts are omitted whenever there is no ambiguity on the referred automaton.*
Similar notations are also used for the other structures we define in the paper.

Often *labelled transition systems* are used as operational models in concurrency. The difference with respect to automata is that in a labelled transition system no initial state is specified. An automaton describes the behavior of a single system, and hence the initial state of the automaton corresponds to the starting point of the system; a labelled

transition system is used to represent the operational semantics of a whole concurrent formalism, and hence an initial state cannot be defined.

Various notions of behavioral preorders and equivalences have been defined on automata. The most important equivalence is *bisimulation equivalence* [Par80, Mil89].

Definition 3 (bisimulation on automata). *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two automata on the same set L of labels. A relation $\mathcal{R} \subseteq Q_1 \times Q_2$ is a* simulation *for $\mathcal{A}_1$ and $\mathcal{A}_2$ if $q_1 \mathcal{R} q_2$ implies:*

for all transitions $t_1 : q_1 \xrightarrow{l} q_1'$ of $\mathcal{A}_1$ there is some transition $t_2 : q_2 \xrightarrow{l} q_2'$ of $\mathcal{A}_2$ such that $q_1' \mathcal{R} q_2'$.

A relation $\mathcal{R} \subseteq Q_1 \times Q_2$ is a bisimulation *for $\mathcal{A}_1$ and $\mathcal{A}_2$ if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are simulations.*
Two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ on the same set of labels are bisimilar, *written $\mathcal{A}_1 \sim \mathcal{A}_2$, if there is some bisimulation $\mathcal{R}$ for $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $q_{01} \mathcal{R} q_{02}$.*

An important result in the theory of automata in concurrency is the existence of *minimal representatives* in the classes of bisimilar automata. Given an automaton, a reduced automaton is obtained by collapsing each class of equivalent states into a single state (and similarly for the transitions). This reduced automaton is bisimilar to the starting one, and any further collapse of states would lead to a non-bisimilar automaton. The reduced automaton is hence "minimal". Moreover, the same minimal automaton (up to isomorphisms) is obtained from bisimilar automata: thus it can be used as a canonical representative of the whole class of bisimilar automata.

In the definition below we denote with $[q]_{\mathcal{R}_\mathcal{A}}$ the class of equivalence of state q with respect to the largest bisimulation equivalence $\mathcal{R}_\mathcal{A}$ on automaton $\mathcal{A}$. With a light abuse of notation, we denote with $[t]_{\mathcal{R}_\mathcal{A}}$ the class of equivalent of transition t, where

$$t_1 \mathcal{R}_\mathcal{A} t_2 \quad \text{iff} \quad s(t_1) \mathcal{R}_\mathcal{A} s(t_2), \quad d(t_1) \mathcal{R}_\mathcal{A} d(t_2) \quad \text{and} \quad o(t_1) = o(t_2).$$

Definition 4 (minimal automata). *The* minimal automaton $\mathcal{A}_{\min}$ *corresponding to automaton $\mathcal{A}$ is defined as follows:*

- $L_{\min} = L$;
- $Q_{\min} = \{[q]_{\mathcal{R}_\mathcal{A}} \mid q \in Q\}$ *and* $T_{\min} = \{[t]_{\mathcal{R}_\mathcal{A}} \mid t \in T\}$;
- $s_{\min}([t]_{\mathcal{R}_\mathcal{A}}) = [s(t)]_{\mathcal{R}_\mathcal{A}}$ *and* $d_{\min}([t]_{\mathcal{R}_\mathcal{A}}) = [d(t)]_{\mathcal{R}_\mathcal{A}}$;
- $o_{\min}([t]_{\mathcal{R}_\mathcal{A}}) = o(t)$;
- $q_{0\min} = [q_0]_{\mathcal{R}_\mathcal{A}}$.

2.2 The π-Calculus

In this section we describe the π-calculus [MPW92, Mil93], a process calculus in which channel names can be used as values in the communications, i.e., channels are first-order values. This possibility of communicating names gives to the π-calculus a rich expressive power: in fact it allows to generate dynamically new channels and to change the interconnection structure of the processes. The π-calculus has been successfully

used to model object oriented languages [Wal95], and also higher-order communications can be easily encoded in the π-calculus [San93a], thus allowing for code migration.

Many versions of π-calculus have appeared in the literature. For simplicity, we consider only the *monadic* π-calculus, and we concentrate on the *ground* variant of its semantics.

Let $\mathcal{N}$ be an infinite, denumerable set of *names*, ranged over by $a, b, \ldots y, z \ldots$, and let *Var* be a finite set of *process identifiers*, denoted by $A, B, \ldots$; the π-calculus (monadic) *processes*, ranged over by $p, q, \ldots$, are defined by the syntax:

$$p ::= \mathbf{0} \mid \pi.p \mid p|p \mid p+p \mid (\nu x)\, p \mid A(x_1, \ldots, x_n)$$

where the *prefixes* π are defined by the syntax:

$$\pi ::= \tau \mid \bar{x}y \mid x(y).$$

The occurrences of y in $x(y).p$ and $(\nu y)\, p$ are bound; *free* and *bound names* of process p are defined as usual and we denote them with $\mathrm{fn}(p)$ and $\mathrm{bn}(p)$ respectively. For each identifier A there is a definition $A(y_1, \ldots, y_n) \stackrel{\text{def}}{=} p_A$ (with y_i all distinct and $\mathrm{fn}(p_A) \subseteq \{y_1, \ldots, y_n\}$); we assume that, whenever A is used, its arity n is respected. Finally we require that each process identifier in p_A is in the scope of a prefix (guarded recursion).

Some comments on the syntax of π-calculus are now in order. As usual, $\mathbf{0}$ is the terminated process. In process $\pi.p$ the prefix π defines an action to execute before p is activated. The prefix $\tau.p$ describes an internal (invisible) action of the process. The *output* prefix $\bar{x}y.p$ specifies the channel x for the communication and the value y that is sent on x. In the *input* prefixes $x(y).p$, name x represents the channel, whereas y is a formal variable: its occurrences in p are instantiated with the received value. Process $p|q$ is the parallel composition with synchronization of p and q, whereas $p+q$ is the nondeterministic choice. Process $(\nu x)\, p$ restricts the possible interactions of process p, disabling communications on channel x.

We use σ, ρ to range over name substitutions, and we denote with $\{y_1/x_1 \cdots y_n/x_n\}$ the substitution that maps x_i into y_i for $i = 1, \ldots, n$ and that is the identity on the other names.

We now introduce a *structural congruence* of π-calculus processes. This structural congruence allows us to identify all the processes which represent essentially the same system and which differ just for syntactical details. The structural congruence $\equiv$ is the smallest congruence which respects the following equivalences

(alpha)	$(\nu x)\, p \equiv (\nu y)\, (p\{y/x\})$ if y does not appear in p
(sum)	$p+\mathbf{0} \equiv p \qquad p+q \equiv q+p \qquad p+(q+r) \equiv (p+q)+r$
(par)	$p\|\mathbf{0} \equiv p \qquad p\|q \equiv q\|p \qquad p\|(q\|r) \equiv (p\|q)\|r$
(res)	$(\nu x)\, \mathbf{0} \equiv \mathbf{0} \qquad (\nu x)\, (\nu y)\, p \equiv (\nu y)\, (\nu x)\, p$
	$(\nu x)\, (p\|q) \equiv p\|(\nu x)\, q$ if x does not appear in p

The structural congruence is useful in practice to obtain finite state representations for classes of processes. It can be used to garbage-collect terminated component — by

exploiting rule $p|\mathbf{0} \equiv p$ — and unused restrictions — by using the rules above, if α does not appear in p then $(\nu\alpha)\, p \equiv p$: in fact, $(\nu x)\, p \equiv (\nu x)\, (p|\mathbf{0}) \equiv p|(\nu x)\, \mathbf{0} \equiv p|\mathbf{0} \equiv p$.

By exploiting the structural congruence $\equiv$, each π-calculus process can be seen as a set of *sequential processes* that act in parallel, sharing a set of channels, some of which are global (unrestricted) while some other are local (restricted). Each sequential process is represented by a term of the form

$$s ::= \pi.p \mid p+p \mid A(x_1, \ldots, x_n)$$

that can be considered as a "program" describing all the possible behaviors of the sequential process.

The *ground* semantics of the π-calculus is the simplest operational semantics that can be defined for this language. It differs from other semantics, such as the *early* and *late* semantics, in the management of input transitions [MPW93]. According to the early semantics, process $x(y).p$ can perform a whole bunch of input transitions

$$x(y).p \xrightarrow{xz} p\{^z/_y\}$$

corresponding to the different names z that the environment can send to the process to instantiate the formal input parameter y. In the ground semantics, instantiation of the input parameters are not taken into account, and process $x(y).p$ can perform only one input transition:

$$x(y).p \xrightarrow{x(y)} p.$$

Ground bisimilarity is easy to check[1]. However, it is less discriminating than early bisimilarity, and does not capture the possibility for the environment of communicating an already existing name during an input transition of a process. For instance,

$$x(y).(\bar{y}y.\mathbf{0}|z(w).\mathbf{0}) \text{ and } x(y).(\bar{y}y.z(w).\mathbf{0} + z(w).\bar{y}y.\mathbf{0})$$

are not equivalent according to the early semantics, since, performing input xz we obtain

$$\bar{z}z.\mathbf{0}|z(w).\mathbf{0} \text{ and } \bar{y}y.z(w).\mathbf{0} + z(w).\bar{y}y.\mathbf{0}$$

and a synchronization (i.e., a τ transition) is possible in the first process but not in the second. However,

$$x(y).(\bar{y}y.\mathbf{0}|z(w).\mathbf{0}) \text{ and } x(y).(\bar{y}y.z(w).\mathbf{0} + z(w).\bar{y}y.\mathbf{0})$$

are equivalent according to the ground semantics since the reception of the already existing name z is not allowed. For simplicity, in this paper we consider only the ground semantics. The presented results, however, can easily be extended to the other π-calculus semantics.

The *ground actions* that a process can perform are defined by the following syntax:

$$\mu ::= \tau \mid x(y) \mid \bar{x}y \mid \bar{x}(y)$$

and are called respectively *synchronization*, *input*, *free output* and *bound output* actions.

[1] ... and, as we will see, easy to model with HD-automata.

Table 1. Free and bound names of π-calculus actions

μ	$\mathrm{fn}(\mu)$	$\mathrm{bn}(\mu)$	$\mathrm{n}(\mu)$
τ	$\emptyset$	$\emptyset$	$\emptyset$
$x(y)$	$\{x\}$	$\{y\}$	$\{x,y\}$
$\overline{x}y$	$\{x,y\}$	$\emptyset$	$\{x,y\}$
$\overline{x}(y)$	$\{x\}$	$\{y\}$	$\{x,y\}$

Table 2. Ground operational semantics of π-calculus

$$[\text{PREF}]\ \pi.p \xrightarrow{\pi} p \qquad [\text{SUM}]\ \frac{p_1 \xrightarrow{\mu} p'}{p_1+p_2 \xrightarrow{\mu} p'}$$

$$[\text{COMM}]\ \frac{p_1 \xrightarrow{\overline{x}y} p_1' \quad p_2 \xrightarrow{x(z)} p_2'}{p_1|p_2 \xrightarrow{\tau} p_1'|(p_2'\{y/z\})} \qquad [\text{PAR}]\ \frac{p_1 \xrightarrow{\mu} p_1'}{p_1|p_2 \xrightarrow{\mu} p_1'|p_2}\ \text{if } \mathrm{bn}(\mu) \cap \mathrm{fn}(p_2) = \emptyset$$

$$[\text{OPEN}]\ \frac{p \xrightarrow{\overline{x}y} p'}{(\nu y)\,p \xrightarrow{\overline{x}(y)} p'}\ \text{if } x \neq y \qquad [\text{CLOSE}]\ \frac{p_1 \xrightarrow{\overline{x}(y)} p_1' \quad p_2 \xrightarrow{x(y)} p_2'}{p_1|p_2 \xrightarrow{\tau} (\nu y)\,(p_1'|p_2')}$$

$$[\text{RES}]\ \frac{p \xrightarrow{\mu} p'}{(\nu x)\,p \xrightarrow{\mu} (\nu x)\,p'}\ \text{if } x \notin \mathrm{n}(\mu)$$

$$[\text{IDE}]\ \frac{p_A\{y_1/x_1 \cdots y_n/x_n\} \xrightarrow{\mu} p'}{A(y_1,\ldots,y_n) \xrightarrow{\mu} p'}\ \text{if } A(x_1,\ldots,x_n) \stackrel{\text{def}}{=} p_A$$

The *free names*, *bound names* and *names* of an action μ, respectively written $\mathrm{fn}(\mu)$, $\mathrm{bn}(\mu)$ and $\mathrm{n}(\mu)$, are defined as in Table 1.

The transitions for the *ground operational semantics* are defined by the axiom schemata and the inference rules of Table 2. We remind that rule

$$\frac{p \equiv p' \quad p' \xrightarrow{\mu} p'' \quad p'' \equiv p''}{p \xrightarrow{\mu} p''}$$

is implicitly assumed.

Notice that the actions a π-calculus process can perform are different from the prefixes. This happens due to the bound output actions. These actions are specific of the π-calculus; they represent the communication of a name that was previously restricted, i.e., it corresponds to the generation of a new channel between the process and the environment: this phenomenon is called *name extrusion*.

Now we present the definition of the ground bisimulation for the π-calculus.

Definition 5 (ground bisimulation). *A relation $\mathcal{R}$ over processes is an* ground simulation *if whenever $p\ \mathcal{R}\ q$ then:*

for each $p \xrightarrow{\mu} p'$ with $\mathrm{bn}(\mu) \cap \mathrm{fn}(p|q) = \emptyset$ there is some $q \xrightarrow{\mu} q'$ such that $p'\ \mathcal{R}\ q'$.

A relation $\mathcal{R}$ is an ground bisimulation *if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are ground simulations.*

Two processes p and q are ground bisimilar, *written $p \sim_g q$, if $p \mathcal{R} q$ for some ground bisimulation $\mathcal{R}$.*

In the definition above, clause "$\mathrm{bn}(\mu) \cap \mathrm{fn}(p|q) = \emptyset$" is necessary to guarantee that the name, that is chosen to represent the newly created channel in a bound output transition, is fresh for both the processes. This clause is necessary since equivalent processes may have different sets of free names.

As for other process calculi, a labelled transition system is used to give an operational semantics to the π-calculus. However, this way to present the operational semantics has some disadvantages. Consider process $q = (\nu y)\,\bar{x}y.y(z).\mathbf{0}$. It is able to generate a new channel by communicating name y in a bound output. The creation of a new name is represented in the transition system by means of an infinite bunch of transitions $q \xrightarrow{\bar{x}(w)} w(z).\mathbf{0}$, where, in this case, w is any name different from x: the creation of a new channel is modeled by using all the names which are not already in use to represent it. As a consequence, the definition of bisimulation is not the ordinary one: in general two bisimilar process can have different sets free names, and the clause "$\mathrm{bn}(\mu) \cap \mathrm{fn}(p|q) = \emptyset$" has to be added in Definition 5 to deal with those bound output transitions which use a name that is used only in one of the two processes. The presence of this clause makes it difficult to reuse standard theory and algorithms for bisimulation on the π-calculus — see for instance [Dam97].

3 History-Dependent Automata

Ordinary automata are successful basic process calculi like CCS. For more sophisticated calculi, however, they are not: in fact, they are not able to capture the particular structures of these languages, that is represented in ordinary automata only in an implicit way. As a consequence, infinite-state automata are often obtained also for very simple programs. To model these languages, it is convenient to enrich states and labels with (part of) the information of the programs, so that the particular structures manipulated by the languages are represented explicitly. These enriched automata are hence more adherent to the languages than ordinary automata.

Different classes of enriched automata can be defined by changing the kind of additional information. Here we focus on a simple form of enriched automata. They are able to manipulate generic "resources": a resource can be allocated, used, and finally released. At this very abstract level, resources can be represented by names: the allocation of a resource is modeled by the generation of a fresh name, that is then used to refer to the resource; since we do not assume any specific operation on resources, the usage of a resource in a transition is modeled by observing the corresponding name in the label; finally, a resource is (implicitly) deallocated when the corresponding name is no more referenced.

We call this class of enriched automata *History-Dependent Automata*, or *HD-automata* in brief. In fact, the usage of names described above can be considered a way to express dependencies between the transitions of the automaton; a transition that uses a name depends on the past transition that generated that name.