Festschrift

LNCS 3525

Ali E. Abdallah
Cliff B. Jones
Jeff W. Sanders (Eds.)

Communicating Sequential Processes

The First 25 Years

Symposium on the Occasion of 25 Years of CSP
London, UK, July 2004
Revised Invited Papers

Springer

Ali E. Abdallah Cliff B. Jones
Jeff W. Sanders (Eds.)

Communicating Sequential Processes

The First 25 Years

Symposium on the Occasion of 25 Years of CSP
London, UK, July 7-8, 2004
Revised Invited Papers

Springer

Volume Editors

Ali E. Abdallah
London South Bank University
Faculty of BCIM
Institute for Computing Research
103 Borough Road, London, SE1 0AA, UK
E-mail: A.Abdallah@lsbu.ac.uk

Cliff B. Jones
University of Newcastle upon Tyne
School of Computing Science
Newcastle upon Tyne, NE1 7RU, UK
E-mail: cliff.jones@ncl.ac.uk

Jeff W. Sanders
Oxford University Computing Laboratory
Parks Road, Oxford OX1 3QD, UK
E-mail: Jeff.Sanders@comlab.ox.ac.uk

The cover illustration is the work of Bill Roscoe.

Library of Congress Control Number: 2005925390

CR Subject Classification (1998): D.2.4, F.3, D.1.3, D.3.1

ISSN	0302-9743
ISBN-10	3-540-25813-2 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-25813-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11423348 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3525

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

Preface

This volume, like the symposium CSP25 which gave rise to it, commemorates the semi-jubilee of Communicating Sequential Processes.

Tony Hoare's paper "Communicating Sequential Processes"[1] is today widely regarded as one of the most influential papers in computer science. To commemorate it, an event was organized under the auspices of BCS-FACS (the British Computer Society's Formal Aspects of Computing Science specialist group). CSP25 was one of a series of such events organized to highlight the use of formal methods, emphasize their relevance to modern computing and promote their wider application. BCS-FACS is proud that Tony Hoare presented his original ideas on CSP at one of its first meetings, in 1978.

The two-day event, 7–8 July 2004, was hosted by London South Bank University's Institute for Computing Research, Faculty of Business, Computing and Information Management. The intention was to celebrate, reflect upon and look beyond the first quarter-century of CSP's contributions to computer science. The meeting examined the impact of CSP on many areas stretching from semantics (mathematical models for understanding concurrency and communications) and logic (for reasoning about behavior), through the design of parallel programming languages (i/o, parallelism, synchronization and threads) to applications varying from distributed software and parallel computing to information security, Web services and concurrent hardware circuits. It included a panel discussion with panelists Brookes, Hoare, de Roever and Roscoe (chaired by Jeff Sanders), poster presentations by PhD students and others, featured a fire alarm (requiring evacuation in the rain!) and concluded with the presentation of a fountain pen to Prof. Sir C. A. R. Hoare.

We owe thanks to the BCS-FACS steering committee and its chairman, Jonathan P. Bowen, for their overwhelming support. Special thanks are due to Dedian Hopkin (LSBU Vice Chancellor) for opening the event in the newly built Keyworth Centre; Chris Clare (Dean), Geoff Elliot (Deputy-Dean), and Terry Fogarty (Head of the Institute for Computing Research) for providing a stimulating environment for hosting the event. Our gratitude goes to our sponsors for their generous support: Microsoft Research, Cambridge, UK; Formal Systems Europe, Limited; Handshake Solutions, Philips, Netherlands; Verified Systems International,GmbH, Germany; Formal Methods Europe (FME); and London South Bank University, Institute for Computing Research. We would like to thank the local organization team: Ali N. Haidar, Michelle Hammond, Kalpesh Kapour and Paul Boca for their hard work to ensure the smooth running of the local arrangements.

[1] *Communications of the ACM*, **21**(8):666–667, 1978.

We would also like to thank Bill Roscoe for his "Golden Valley"[2] painting used in the cover of this book. This was the favorite among CSP25 authors who considered several other alternatives. What's its relevance to CSP25? In the words of one of the contributing authors:

> It's a lovely scene with a prominent feature, the much-branching tree representing CSP, and the road winding off representing the 25 years so far, with the rest hidden behind the tree. (Who knows where it may still lead?)

After presentation at the symposium, the contributions were reworked by their authors and fully refereed. We are grateful to all for their timely and efficient work, particularly in the refereeing process where helpful and incisive comments were made. The resulting papers are gathered here, as they were in the workshop, into session-sized chunks, described below.

The conference website can be found at www.lsbu.ac.uk/menass/csp25 and www.bcs-facs.org

Semantic Foundations

The first paper to confront the denotational semantics of CSP with due regard to the interplay between communication and abstraction was "A Theory of Communicating Sequential Processes[3]" by Steve Brookes, Tony Hoare and Bill Roscoe. Before it, the simplistic but intuitively compelling traces model had been the basis for a semantics capable of capturing safety properties but not of capturing liveness (being too weak to capture deadlock or divergence). That paper concentrated on the communicating fragment of CSP, TCSP, based on recursively defined communicating processes evolving in parallel. The study concentrated, inevitably, on the distinction between the choice of events due to the environment choosing from a menu (external choice) and as the result of abstraction (internal, or nondeterministic, choice); in a subsequent paper[4] Brookes and Roscoe extended the denotational model to account also for divergence.

This collection begins with two papers on the semantic foundations of CSP. Brookes replaces the naive traces semantics with one based on actions and Roscoe extends the semantics of divergence and provides an appropriate definition of fixed point. Each paper responds to developments in theoretical computer science during the couple of decades since the 1984 and 1985 papers: the former by acknowledging work on action-based transition systems and the latter by acknowledging progress in our understanding of divergence and fixed points. Each paper provides stronger techniques whilst retaining the flavor of the original CSP.

---

[2] Autumn scene in the Golden Valley, Herefordshire, 2000.

[3] JACM, 31(3):560–599, 1984.

[4] "An Improved Failures Model for CSP," Proc. Seminar on Concurrency, Springer, LNCS 197, 1985.

In the cleverly titled *Retracing the Semantics of CSP* Brookes argues for a traces semantics that is at once more general than that of CSP and yet retains much of the simplicity and design elegance of the original. The only cost is re-evaluation of the notion of trace, to make it action based, and imposition of a fairness condition on processes. The result is a general formalism allowing a bisimulation-type equivalence between processes that differ only in atomicity of their actions.

In *Seeing Beyond Divergence* Roscoe shows how to refine the standard denotational model of a mild extension of TCSP to reveal traces of a process, more extended than just the minimal traces, after which it may diverge. Concentrating on possible divergence, and so ignoring 'refusals or failures' information, he constructs a model (named $\mathcal{SBD}$ as in the title of the paper) to distinguish a process's various opportunities to diverge — something TCSP has never done. To provide meaning to recursion in $\mathcal{SBD}$ Roscoe shows that neither greatest nor least fixed points would be correct and so he is forced to use a two-stage process whose result he calls a reflected fixed point.

A further contribution to fixed-point theory in CSP is provided by Mike Reed in his paper *Order, Topology and Recursion Induction in CSP* later in this volume.

Refinement and Simulation

The major difference between CSP and, for example, the process algebra CCS[5] lies in the distinction each makes between processes. Whilst processes in CSP are related by refinement (one can be replaced by the other for the purpose of implementation), those in CCS are related by the finer notion of (bi)simulation.

In July 2002 a workshop was held at Microsoft Research Ltd. Cambridge to contemplate the differences and similarities between the various process algebras, with the aim of reconciling the fundamental ideas of refinement and simulation, particularly for CSP and CCS. One outcome has been the two papers in this section. Not surprisingly, established co-authors Hoare and He have produced related treatments. Each paper uses the notion of barbed traces in a treatment of process algebra in which refinement and simulation coincide. In fact a combination of the papers, which the reader will find of quite contrasting styles, might be regarded as an extra chapter for their book[6]. Use of barbed traces might be regarded as an alternative solution to the high-level plan of Brookes.

Hardware Synthesis

From its early days CSP has been closely associated with hardware design. David May provides an entertaining account of those days in *CSP,* occam *and transputers*, the paper of his after-dinner speech. He makes a convincing case for

[5] Robin Milner, *Communication and Concurrency*, Prentice-Hall, 1989.
[6] *Unifying Theories of Programming*, Prentice-Hall, 1998.

remembering our own (collective) principles as we progress and for valuing more highly the things at which we are good; and he draws potent conclusions for industry, research and education.

At the same time as inmos, occam and CSP exploited highly-synchronized communication, asynchronous hardware design was enjoying a resurgence of popularity[7]. The appropriate modification to CSP and the revised laws (thought of as being obtained by inserting unbounded buffers along channels) was undertaken by Mark Josephs whose paper *Models for Data-Flow Sequential Processes* extends that work to a wider family of processes and more sophisticated semantic models. It provides some laws and concentrates on denotational semantics.

Philips Electronics, Eindhoven, has shown a long-standing commitment to the use of formal methods and in particular to the work on asynchronous CSP. In his monumental paper *Implementation of Handshake Components*, Ad Peeters shows how CSP underlies the techniques of the established Handshake Technology developed at Philips for the design and implementation of unclocked circuits. The interest focuses on handshaking protocols that are efficient and correct in the various paradigms for unclocked design — summarized in this self-contained article. Peeters demonstrates the remarkable extent to which process algebra successfully pervades the various levels of abstraction.

Transactions

The laws satisfied by asynchronous processes communicating lazily via streams, as treated in the previous section, for example in the article by Mark Josephs, resemble those satisfied in transaction processing: a topic at the heart of applied formal methods. In fact, in his book *Communicating Sequential Processes* Hoare introduced operators to model the interrupt, checkpoint, rollback and recovery of transaction-processing systems. In this section that topic is further explored; the main concern is to maintain atomicity in a distributed system. Some treatments have attempted to do so using event refinement, the process algebra version of the data refinement of sequential programming.

But in *A Trace Semantics for Long-Running Transactions* Michael Butler, Tony Hoare and Carla Ferreira give an elegant calculus of compensations for a restriction of CSP to achieve a similar result. They adopt a traces semantics in which an action is compensable if it can subsequently be undone atomically, and presents a compositional 'cancellation' semantics for processes with nested interruption and compensation.

In *Practical Application of CSP and FDR to Software Design* Jonathan Lawrence acknowledges the difficulty confronting transfer of research — in this case concerning CSP — to industry and presents a case study encapsulating valuable lessons. The study centers on a recent IBM project using CSP and FDR to produce a multi-threaded connection pooling mechanism connecting a transaction-processing system to a Web server. The project spanned three days

[7] Ivan Sutherland, "Micropipelines", CACM, 32:720–738, 1989.

and included formal specification in CSP of the required system, validation with some degree of confidence that it captured the informal requirements, expression of the design in CSP and verification of its correctness in FDR. The result was so successful that subsequent enhancements to the delivered Java code could confidently be done by hand. Lawrence highlights the value of applied MSc's which include projects providing students with an opportunity to transfer what they have learnt on the MSc to the workplace.

Concurrent Programming

The extremely active occam user group continues the application of CSP begun in the work described by May in this volume to programming-language design. In *Communicating Mobile Processes* Peter Welch and Frederick Barnes introduce occam-π as a hybrid of occam and the π-calculus introduced by Milner and studied extensively in CCS. The approach is largely pragmatic, including benchmarks and the outline of applications. It is envisaged that a semantics would be denotational, following those of CSP and influenced by the π-calculus.

In *Model-Based Design of Concurrent Programs* Jeff Magee and Jeff Kramer use Label Transition Systems (LTS), a notation based on CSP, to model concurrent systems and to study their behaviours. Their approach combines clear modelling with tools that support graphical animations and systematic generation of parallel implementations in concurrent Java. Both safety checks (essentially traces properties) and liveness checks (under assumptions concerning scheduling and choice) are achieved. They conclude that such animations are useful both to students and practitioners in overcoming resistance to formal methods.

Security

One of formal methods' huge successes in the past decade has been to reasoning about security. In terms of CSP, the success has been largely due to work by Bill Roscoe et al. and Gavin Lowe (with the Caspar tool).

In *Verifying Security Protocols: an Application of CSP* Steve Schneider and Rob Delicata provide an elegant case study showing how CSP, with the notion of a rank function, can be used to reason about an authentication protocol. After proposing a putative protocol their analysis locates a flaw and verifies the correctness of a modification. In verification, the rank function is used to show that illegitimate messages do not occur. The paper is self-contained and might be used by those familiar with CSP as an introduction to this topical area.

Over the years various models of computation have been used to formalize non-interference. Typically these floundered on non-determinism, "input/output" distinctions, input totality and so forth. In *Shedding Light on Haunted Corners of Information Security* Peter Ryan outlines how process algebras, in particular CSP, can be applied to give a formal characterization of the absence of information flows through a system. Unfortunately, Peter Ryan was unable to attend due to compelling personal reasons at the last minute. Hence, only the abstract of his talk is included in this volume.

Linking Theories

Whilst security has provided one important playing field for CSP, probability has provided another. The challenge is to express and reason about distributed probabilistic algorithms using a variant of process algebra that includes a combinator for choice, with given probability, between two processes. Unacceptable attempts abound. In *Of Probabilistic wp and CSP — and Compositionality*, Carroll Morgan starts 'afresh' from the successful work on probabilistic sequential programming and targets process algebra via the intermediary of action systems. His translation throws up healthiness conditions for probabilistic CSP and suggests a program of work that might — finally — result in a compositional probabilistic process algebra. Incidentally his discussion of (general) compositionality using the example of eye color and the Mendelian concept of allele is a gem.

In this section is included the abstract for the talk by Mike Reed *Order, Topology and Recursion Induction in CSP* that might be thought of as a contribution to semantic foundations. He presents a recursion-induction principle that produces least fixed points for functions whose least fixed points are maximal (i.e., deterministic in the failures model of CSP). The setting is a Scott domain and the results are general enough to cover existing instances of recursion induction in CSP; in topology they are strong enough to provide answers to open questions from domain theory and point-set topology.

Automated Development, Reasoning and Model Checking

As a formal method, CSP was slow to respond to the pressure for automation. Perhaps as a result, Formal Methods' tool FDR achieved immediate success; for instance it has played a crucial role in many of the papers in this volume. But it, and its scope, still progress as the papers by Michael Goldsmith and Ranko Lazić indicate.

In *Operational Semantics for Fun and Profit* Michael Goldsmith observes that a source of computation inefficiency in FDR is evaluation of the structured operational semantics of the operationally–presented target system (an evaluation that is necessary whenever a denotational property is to be determined). He proposes a *supercompilation* procedure to overcome it, if not in every case then at least in many. An unexpected benefit of supercompilation is transformation of a process to a form accessible to previously studied *watchdog* transformations that enable a refinement check to be recast in more efficient form.

The method of data independence allows a model-checking argument, concerning a process whose data type takes on a single value, to be extended to that process with arbitrary data value. In *On model Checking Data-Independent Systems with Arrays with Whole-Array Operations* Ranko Lazić, the originator of the technique of data independence in CSP, Tom Newcomb, and Bill Roscoe show how to extend it to programs using arrays indexed by one data-independent variable that have values from another. They obtain simple and natural conditions for decidability or undecidability of realistic questions concerning the use of such types.

For all its use, and all its appearance in this volume, FDR is far from being the only formalism for animating CSP. In the article by Magee and Kramer *Model-Based Design of Concurrent Programs* an alternative has already been demonstrated.

Industrial-Strength CSP

We have seen how CSP has been used to study theoretical aspects of concurrency and that it seems to offer yet further potential for doing so. We have seen how it has been used in hardware design, at both the implementation and design levels. And we have seen how its tools offer industrial-strength model checking. But what about the broader scope of software engineering?

In *Industrial-Strength CSP: Opportunities and Challenges in Model-Checking*, Sadie Creese demonstrates the use of FDR in reasoning about various aspects of high-integrity systems from industry, as seen from her perspective in the Systems Assurance Group within QinetiQ.

In the paper *Applied Formal Methods — from CSP to Executable Hybrid Specifications* Jan Peleska discusses his work at Verified Systems International and the University of Bremen. His case studies are drawn from an impressively realistic range, including an implementation of Byzantine agreement to provide a fault-tolerant component of the International Space Station, and the avionics controller of the Airbus A340. He discusses the difficulties involved in the production of large and complex systems. Hybrid methods become important and executability, in the form of tools available for prototyping, necessary to convince coworkers. But in the end formal methods, and in particular CSP, remainS just one of a spectrum of techniques that contribute to product quality.

Reflections!

It is not often that burgeoning areas are afforded the luxury of reflecting on both their past and futures. With the contributions contained in this volume the reader has evidence enough to decide the relevance of Gilbert Ryle's warning (*Dilemmas*, The Tarner Lectures, 1953, Cambridge University Press, digital printing 2002, page 14.)

> Karl Marx was sapient enough to deny the impeachment that he was a Marxist. So too Plato was, in my view, a very unreliable Platonist. He was too much of a philosopher to think that anything that he had said was the last word. It was left to his disciples to identify his foot marks with his destination.

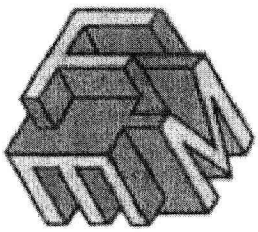
Ali E. Abdallah, Cliff B. Jones and Jeff W. Sanders
London, Newcastle and Oxford, January 2005

Sponsors

BCS- Formal Aspect of Computing Science specialist group

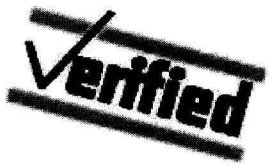
London South Bank University, UK

Microsoft Research, Cambridge, UK



Formal Methods Europe

Handshake Solutions, Philips, Netherlands



Verified Systems International, GmbH, Germany

Formal Systems (Europe) Limited

Lecture Notes in Computer Science

For information about Vols. 1–3362

please contact your bookseller or Springer

Vol. 3525: A.E. Abdallah, C.B. Jones, J.W. Sanders (Eds.), Communicating Sequential Processes. XIV, 321 pages. 2005.

Vol. 3492: P. Blache, E. Stabler, J. Busquets, R. Moot (Eds.), Logical Aspects of Computational Linguistics. X, 363 pages. 2005. (Subseries LNAI).

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. 3463: M. Dal Cin, M. Kaâniche, A. Pataricza (Eds.), Dependable Computing - EDCC 2005. XVI, 472 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 (Eds.), 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 II. 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 (Ed.), 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.

Table of Contents

Linking Theories

Security

Automated Development and Model Checking

Industrial-Strength CSP

Retracing the Semantics of CSP

Stephen Brookes

Carnegie Mellon University

Abstract. CSP was originally introduced as a parallel programming language in which sequential imperative processes execute concurrently and communicate by synchronized input and output. The influence of CSP and the closely related process algebra TCSP is widespread. Over the years CSP has been equipped with a series of denotational semantic models, involving notions such as communication traces, failure sets, and divergence traces, suitable for compositional reasoning about safety properties and deadlock analysis. We revisit these notions (and review some of the underlying philosophy) with the benefit of hindsight, and we introduce a semantic framework based on action traces that permits a unified account of shared memory parallelism, asynchronous communication, and synchronous communication. The framework also allows a relatively straightforward account of (a weak form of) fairness, so that we obtain models suitable for compositional reasoning about liveness properties as well as about safety properties and deadlock. We show how to incorporate race detection into this semantic framework, leading to models more independent of hardware assumptions about the granularity of atomic actions.

1 Introduction

The parallel programming language CSP was introduced in Tony Hoare's classic paper [15]. As originally formulated, CSP is an imperative language of guarded commands [11], extended with primitives for input and output and a form of parallel composition which permits synchronized communication between named processes. The original language derives its full name from the built-in syntactic constraint that processes belong to the sequential subset of the language. The syntax of programs was also constrained to preclude concurrent attempts by one process to write to a variable being used by another process: this may be expressed succinctly as the requirement that processes have "disjoint local states". These design decisions, influenced by Dijkstra's principle of "loose coupling" [10], lead to an elegant programming language in which processes interact solely by message-passing. Ideas from CSP have passed the test of time, having influenced the design of more recent parallel programming languages such as Ada, occam [18], and Concurrent ML [26].

Most of the subsequent foundational research has focussed on a process algebra known as *Theoretical CSP* (or *TCSP*) in which the imperative aspects of the original language are suppressed [2]. In TCSP (and in occam) processes

A.E. Abdallah, C.B. Jones, and J.W. Sanders (Eds.): CSP25, LNCS 3525, pp. 1–14, 2005.