Marcello M. Bonsangue
Einar Broch Johnsen (Eds.)

LNCS 4468

Formal Methods for Open Object-Based Distributed Systems

9th IFIP WG 6.1 International Conference FMOODS 2007
Paphos, Cyprus, June 2007
Proceedings

ifip

Springer

Marcello M. Bonsangue
Einar Broch Johnsen (Eds.)

Formal Methods for Open Object-Based Distributed Systems

9th IFIP WG 6.1
International Conference FMOODS 2007
Paphos, Cyprus, June 6-8, 2007
Proceedings

Springer

Volume Editors

Marcello M. Bonsangue
Leiden University
Leiden Institute of Advanced Computer Science
2300 RA Leiden, The Netherlands
E-mail: marcello@liacs.nl

Einar Broch Johnsen
University of Oslo
Department of Informatics
PO Box 1080 Blindern, 0316 Oslo, Norway
E-mail: einarj@ifi.uio.no

Library of Congress Control Number: 2007927619

CR Subject Classification (1998): C.2.4, D.1.3, D.2, D.3, F.3, D.4

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN	0302-9743
ISBN-10	3-540-72919-4 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-72919-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12072873 06/3180 5 4 3 2 1 0

Lecture Notes in Computer Science 4468

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

Preface

This volume contains the proceedings of the Ninth IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2007). The conference is part of the federated conferences on Distributed Computing Techniques (DisCoTec), together with the Ninth International Conference on Coordination Models and Languages (COORDINATION 2007) and the Seventh IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2007). It was organized by the Department of Computer Science of the University of Cyprus, and it was held in Paphos, Cyprus during June, 6-8, 2007. The event was the third federated DisCoTec conference series, initiated in Athens in June 2005 and continued in Bologna in June 2006.

The goal of the FMOODS conferences is to bring together researchers and practitioners whose work encompasses three important and related fields:

- Formal methods
- Distributed systems and
- Object-based technology

The 17 papers presented at FMOODS 2007 and included in this volume were selected by the Programme Committee among 45 submissions. Each submission was reviewed by at least three Programme Committee members. They all reflect the scope of the conference and cover the following topics: semantics of object-oriented programming; formal techniques for specification, analysis, and refinement; model checking; theorem proving and deductive verification; type systems and behavioral typing; formal methods for service-oriented computing; integration of quality of service requirements into formal models; formal approaches to component-based design; and applications of formal methods.

The two invited speakers of FMOODS 2007, who also contributed to the proceedings in this volume, were:

- Mariangiola Dezani-Ciancaglini of the University of Torino, Italy, who is well known for her work on intersection-type assignment systems, which were largely used as finitary descriptions of lambda-models. More recently, she is interested in type systems for object-oriented languages and ambient calculi. In this volume she contributed with a paper introducing session types for object-oriented languages.
- Wolfgang Ahrend of the Chalmers Technical University, Sweden, who is known for his involvement in the KeY project, which aims at a formal methods tool that integrates design, implementation, formal specification, and formal verification of object-oriented software.

The conference was supported by IFIP, in particular by TC 6 and the working group WG 6.1. Thanks are due to John Derrick, Elie Najm, and George

Papadopoulos for their efforts in this respect. We are also grateful to the University of Cyprus and George Papadopoulos for organizing the event, the European project IST-33826 CREDO (`http://credo.cwi.nl`) for sponsoring one invited speaker, and the ITEA project Trust4All for sponsoring the participation of one of the Programme Committee Chairs.

Finally we thank all authors for the high quality of their contributions, and the Programme Committee members and the external reviewers for their help in selecting the papers for this volume. The use of EasyChair as a conference system was very helpful for organizing the technical programme and proceedings of FMOODS 2007.

June 2007

Marcello Bonsangue
Einar Broch Johnsen

Conference Organization

Programme Chairs

Marcello Bonsangue
Einar Broch Johnsen

Programme Committee

Bernhard Aichernig
Alessandro Aldini
Frank de Boer
Eerke Boiten
John Derrick
Robert France
Reiko Heckel
Naoki Kobayashi
Zhiming Liu
Elie Najm
David Naumann
Uwe Nestmann
Erik Poll
Antonio Ravara
Arend Rensink
Ralf Reussner
Grigore Roşu
Bernhard Rumpe
Martin Steffen
Carolyn Talcott
Heike Wehrheim
Martin Wirsing
Wang Yi
Gianluigi Zavattaro
Elena Zucca

Local Organization

George Papadopoulos

External Reviewers

Lacramioara Astefanoaei
Khaled Barbaria
Steffen Becker
Dénes Bisztray
Laura Bocchi
Mario Bravetti
Jan Broersen
Marzia Buscemi
Nadia Busi
Maura Cerioli
Feng Chen
Hung Dang Van
Alessandra Di Pierro
Sonia Fagorzi
Boris Gajanovic
Hans Groenniger
Claudio Guidi
Christian Haack
Christoph Herrmann
Mark Hills
Mohammad Mahdi Jaghoori
Jens Happe
Klaus Krogmann
Ruurd Kuiper
Marcel Kyas
Giovanni Lagorio
Xiaoshan Li
Francisco Martins
Björn Metzler
Roland Meyer
Yasuhiko Minamide
Kazuhiro Ogata
Luca Padovani
Razvan Popescu
Jaime Ramos
Gianna Reggio
Dirk Reiss
Birna van Riemsdijk
Ismael Rodriguez
Martin Schindler
Rudolf Schlatte
Marvin Schulze-Quester
Traian Florin Şerbănuţă
Frank Stomp
Sameer Sundresh
Vasco T. Vasconcelos
Naijun Zhan

Lecture Notes in Computer Science

For information about Vols. 1–4414

please contact your bookseller or Springer

Vol. 4534: I. Tomkos, F. Neri, J. Solé Pareta, X. Masip Bruin, S. Sánchez Lopez (Eds.), Optical Network Design and Modeling. XI, 460 pages. 2007.

Vol. 4531: J. Indulska, K. Raymond (Eds.), Distributed Applications and Interoperable Systems. XI, 337 pages. 2007.

Vol. 4526: M. Malek, M. Reitenspieß, A. van Moorsel (Eds.), Service Availability. X, 155 pages. 2007.

Vol. 4525: C. Demetrescu (Ed.), Experimental and Efficient Algorithms. XIII, 448 pages. 2007.

Vol. 4523: Y.-H. Lee, H.-N. Kim, J. Kim, Y. Park, L.T. Yang, S.W. Kim (Eds.), Embedded Software and Systems. XIX, 829 pages. 2007.

Vol. 4521: J. Katz, M. Yung (Eds.), Applied Cryptography and Network Security. XIII, 498 pages. 2007.

Vol. 4519: E. Franconi, M. Kifer, W. May (Eds.), The Semantic Web: Research and Applications. XVIII, 830 pages. 2007.

Vol. 4517: F. Boavida, E. Monteiro, S. Mascolo, Y. Koucheryavy (Eds.), Wired/Wireless Internet Communications. XIV, 382 pages. 2007.

Vol. 4515: M. Naor (Ed.), Advances in Cryptology - EUROCRYPT 2007. XIII, 591 pages. 2007.

Vol. 4514: S.N. Artemov, A. Nerode (Eds.), Logical Foundations of Computer Science. XI, 513 pages. 2007.

Vol. 4510: P. Van Hentenryck, L. Wolsey (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. X, 391 pages. 2007.

Vol. 4509: Z. Kobti, D. Wu (Eds.), Advances in Artificial Intelligence. XII, 552 pages. 2007. (Sublibrary LNAI).

Vol. 4506: D. Zeng, I. Gotham, K. Komatsu, C. Lynch, M. Thurmond, D. Madigan, B. Lober, J. Kvach, H. Chen (Eds.), Intelligence and Security Informatics: Biosurveillance. XI, 234 pages. 2007.

Vol. 4504: J. Huang, R. Kowalczyk, Z. Maamar, D. Martin, I. Müller, S. Stoutenburg, K.P. Sycara (Eds.), Service-Oriented Computing: Agents, Semantics, and Engineering. X, 175 pages. 2007.

Vol. 4501: J. Marques-Silva, K.A. Sakallah (Eds.), Theory and Applications of Satisfiability Testing – SAT 2007. XI, 384 pages. 2007.

Vol. 4500: N. Streitz, A. Kameas, I. Mavrommati (Eds.), The Disappearing Computer. XVIII, 304 pages. 2007.

Vol. 4496: N.T. Nguyen, A. Grzech, R.J. Howlett, L.C. Jain (Eds.), Agent and Multi-Agent Systems: Technologies and Applications. XXI, 1046 pages. 2007. (Sublibrary LNAI).

Vol. 4493: D. Liu, S. Fei, Z. Hou, H. Zhang, C. Sun (Eds.), Advances in Neural Networks – ISNN 2007, Part III. XXVI, 1215 pages. 2007.

Vol. 4492: D. Liu, S. Fei, Z. Hou, H. Zhang, C. Sun (Eds.), Advances in Neural Networks – ISNN 2007, Part II. XXVII, 1321 pages. 2007.

Vol. 4491: D. Liu, S. Fei, Z.-G. Hou, H. Zhang, C. Sun (Eds.), Advances in Neural Networks – ISNN 2007, Part I. LIV, 1365 pages. 2007.

Vol. 4490: Y. Shi, G.D. van Albada, J. Dongarra, P.M.A. Sloot (Eds.), Computational Science – ICCS 2007, Part IV. XXXVII, 1211 pages. 2007.

Vol. 4489: Y. Shi, G.D. van Albada, J. Dongarra, P.M.A. Sloot (Eds.), Computational Science – ICCS 2007, Part III. XXXVII, 1257 pages. 2007.

Vol. 4488: Y. Shi, G.D. van Albada, J. Dongarra, P.M.A. Sloot (Eds.), Computational Science – ICCS 2007, Part II. XXXV, 1251 pages. 2007.

Vol. 4487: Y. Shi, G.D. van Albada, J. Dongarra, P.M.A. Sloot (Eds.), Computational Science – ICCS 2007, Part I. LXXXI, 1275 pages. 2007.

Vol. 4486: M. Bernardo, J. Hillston (Eds.), Formal Methods for Performance Evaluation. VII, 469 pages. 2007.

Vol. 4485: F. Sgallari, A. Murli, N. Paragios (Eds.), Scale Space Methods and Variational Methods in Computer Vision. XV, 931 pages. 2007.

Vol. 4484: J.-Y. Cai, S.B. Cooper, H. Zhu (Eds.), Theory and Applications of Models of Computation. XIII, 772 pages. 2007.

Vol. 4483: C. Baral, G. Brewka, J. Schlipf (Eds.), Logic Programming and Nonmonotonic Reasoning. IX, 327 pages. 2007. (Sublibrary LNAI).

Vol. 4482: A. An, J. Stefanowski, S. Ramanna, C.J. Butz, W. Pedrycz, G. Wang (Eds.), Rough Sets, Fuzzy Sets, Data Mining and Granular Computing. XIV, 585 pages. 2007. (Sublibrary LNAI).

Vol. 4481: J. Yao, P. Lingras, W.-Z. Wu, M. Szczuka, N.J. Cercone, D. Ślęzak (Eds.), Rough Sets and Knowledge Technology. XIV, 576 pages. 2007. (Sublibrary LNAI).

Vol. 4480: A. LaMarca, M. Langheinrich, K.N. Truong (Eds.), Pervasive Computing. XIII, 369 pages. 2007.

Vol. 4479: I.F. Akyildiz, R. Sivakumar, E. Ekici, J.C.d. Oliveira, J. McNair (Eds.), NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet. XXVII, 1252 pages. 2007.

Vol. 4478: J. Martí, J.M. Benedí, A.M. Mendonça, J. Serrat (Eds.), Pattern Recognition and Image Analysis, Part II. XXVII, 657 pages. 2007.

Vol. 4477: J. Martí, J.M. Benedí, A.M. Mendonça, J. Serrat (Eds.), Pattern Recognition and Image Analysis, Part I. XXVII, 625 pages. 2007.

Vol. 4476: V. Gorodetsky, C. Zhang, V.A. Skormin, L. Cao (Eds.), Autonomous Intelligent Systems: Multi-Agents and Data Mining. XIII, 323 pages. 2007. (Sublibrary LNAI).

Vol. 4475: P. Crescenzi, G. Prencipe, G. Pucci (Eds.), Fun with Algorithms. X, 273 pages. 2007.

Vol. 4472: M. Haindl, J. Kittler, F. Roli (Eds.), Multiple Classifier Systems. XI, 524 pages. 2007.

Vol. 4471: P. Cesar, K. Chorianopoulos, J.F. Jensen (Eds.), Interactive TV: a Shared Experience. XIII, 236 pages. 2007.

Vol. 4470: Q. Wang, D. Pfahl, D.M. Raffo (Eds.), Software Process Dynamics and Agility. XI, 346 pages. 2007.

Vol. 4468: M.M. Bonsangue, E.B. Johnsen (Eds.), Formal Methods for Open Object-Based Distributed Systems. X, 317 pages. 2007.

Vol. 4465: T. Chahed, B. Tuffin (Eds.), Network Control and Optimization. XIII, 305 pages. 2007.

Vol. 4464: E. Dawson, D.S. Wong (Eds.), Information Security Practice and Experience. XIII, 361 pages. 2007.

Vol. 4463: I. Măndoiu, A. Zelikovsky (Eds.), Bioinformatics Research and Applications. XV, 653 pages. 2007. (Sublibrary LNBI).

Vol. 4462: D. Sauveron, K. Markantonakis, A. Bilas, J.-J. Quisquater (Eds.), Information Security Theory and Practices. XII, 255 pages. 2007.

Vol. 4459: C. Cérin, K.-C. Li (Eds.), Advances in Grid and Pervasive Computing. XVI, 759 pages. 2007.

Vol. 4453: T. Speed, H. Huang (Eds.), Research in Computational Molecular Biology. XVI, 550 pages. 2007. (Sublibrary LNBI).

Vol. 4452: M. Fasli, O. Shehory (Eds.), Agent-Mediated Electronic Commerce. VIII, 249 pages. 2007. (Sublibrary LNAI).

Vol. 4451: T.S. Huang, A. Nijholt, M. Pantic, A. Pentland (Eds.), Artifical Intelligence for Human Computing. XVI, 359 pages. 2007. (Sublibrary LNAI).

Vol. 4450: T. Okamoto, X. Wang (Eds.), Public Key Cryptography – PKC 2007. XIII, 491 pages. 2007.

Vol. 4448: M. Giacobini et al. (Ed.), Applications of Evolutionary Computing. XXIII, 755 pages. 2007.

Vol. 4447: E. Marchiori, J.H. Moore, J.C. Rajapakse (Eds.), Evolutionary Computation,Machine Learning and Data Mining in Bioinformatics. XI, 302 pages. 2007.

Vol. 4446: C. Cotta, J. van Hemert (Eds.), Evolutionary Computation in Combinatorial Optimization. XII, 241 pages. 2007.

Vol. 4445: M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, A.I. Esparcia-Alcázar (Eds.), Genetic Programming. XI, 382 pages. 2007.

Vol. 4444: T. Reps, M. Sagiv, J. Bauer (Eds.), Program Analysis and Compilation, Theory and Practice. X, 361 pages. 2007.

Vol. 4443: R. Kotagiri, P.R. Krishna, M. Mohania, E. Nantajeewarawat (Eds.), Advances in Databases: Concepts, Systems and Applications. XXI, 1126 pages. 2007.

Vol. 4440: B. Liblit, Cooperative Bug Isolation. XV, 101 pages. 2007.

Vol. 4439: W. Abramowicz (Ed.), Business Information Systems. XV, 654 pages. 2007.

Vol. 4438: L. Maicher, A. Sigel, L.M. Garshol (Eds.), Leveraging the Semantics of Topic Maps. X, 257 pages. 2007. (Sublibrary LNAI).

Vol. 4433: E. Şahin, W.M. Spears, A.F.T. Winfield (Eds.), Swarm Robotics. XII, 221 pages. 2007.

Vol. 4432: B. Beliczynski, A. Dzielinski, M. Iwanowski, B. Ribeiro (Eds.), Adaptive and Natural Computing Algorithms, Part II. XXVI, 761 pages. 2007.

Vol. 4431: B. Beliczynski, A. Dzielinski, M. Iwanowski, B. Ribeiro (Eds.), Adaptive and Natural Computing Algorithms, Part I. XXV, 851 pages. 2007.

Vol. 4430: C.C. Yang, D. Zeng, M. Chau, K. Chang, Q. Yang, X. Cheng, J. Wang, F.-Y. Wang, H. Chen (Eds.), Intelligence and Security Informatics. XII, 330 pages. 2007.

Vol. 4429: R. Lu, J.H. Siekmann, C. Ullrich (Eds.), Cognitive Systems. X, 161 pages. 2007. (Sublibrary LNAI).

Vol. 4427: S. Uhlig, K. Papagiannaki, O. Bonaventure (Eds.), Passive and Active Network Measurement. XI, 274 pages. 2007.

Vol. 4426: Z.-H. Zhou, H. Li, Q. Yang (Eds.), Advances in Knowledge Discovery and Data Mining. XXV, 1161 pages. 2007. (Sublibrary LNAI).

Vol. 4425: G. Amati, C. Carpineto, G. Romano (Eds.), Advances in Information Retrieval. XIX, 759 pages. 2007.

Vol. 4424: O. Grumberg, M. Huth (Eds.), Tools and Algorithms for the Construction and Analysis of Systems. XX, 738 pages. 2007.

Vol. 4423: H. Seidl (Ed.), Foundations of Software Science and Computational Structures. XVI, 379 pages. 2007.

Vol. 4422: M.B. Dwyer, A. Lopes (Eds.), Fundamental Approaches to Software Engineering. XV, 440 pages. 2007.

Vol. 4421: R. De Nicola (Ed.), Programming Languages and Systems. XVII, 538 pages. 2007.

Vol. 4420: S. Krishnamurthi, M. Odersky (Eds.), Compiler Construction. XIV, 233 pages. 2007.

Vol. 4419: P.C. Diniz, E. Marques, K. Bertels, M.M. Fernandes, J.M.P. Cardoso (Eds.), Reconfigurable Computing: Architectures, Tools and Applications. XIV, 391 pages. 2007.

Vol. 4418: A. Gagalowicz, W. Philips (Eds.), Computer Vision/Computer Graphics Collaboration Techniques. XV, 620 pages. 2007.

Vol. 4416: A. Bemporad, A. Bicchi, G. Buttazzo (Eds.), Hybrid Systems: Computation and Control. XVII, 797 pages. 2007.

Vol. 4415: P. Lukowicz, L. Thiele, G. Tröster (Eds.), Architecture of Computing Systems - ARCS 2007. X, 297 pages. 2007.

¥732.00元

Table of Contents

Invited Talks

Model Checking

Rewriting Logic

Components and Services

Algebraic Calculi

Specification, Verfication and Refinenment

Quality of Service

Asynchronous Session Types and Progress for Object Oriented Languages*

Mario Coppo[1], Mariangiola Dezani-Ciancaglini[1], and Nobuko Yoshida[2]

[1] Dipartimento di Informatica, Università di Torino
{coppo,dezani}@di.unito.it
[2] Department of Computing, Imperial College London
yoshida@doc.ic.ac.uk

Abstract. A session type is an abstraction of a sequence of heterogeneous values sent over one channel between two communicating processes. Session types have been introduced to guarantee consistency of the exchanged data and, more recently, *progress* of the session, i.e. the property that once a communication has been established, well-formed programs will never starve at communication points. A relevant feature which influences progress is whether the communication is synchronous or asynchronous. In this paper, we first formulate a typed asynchronous multi-threaded object-oriented language with thread spawning, iterative and higher order sessions. Then we study its progress through a new effect system. As far as we know, ours is the first session type system which assures progress in asynchronous communication.

1 Introduction

Distributed and concurrent programming paradigms are increasingly interesting, owing to the huge amount of distributed applications and services spread on the Internet. This gives a strong motivation to the study of specifications and implementations of these programs together with techniques for the formal verification of their properties. One of the crucial aspects is that of protocol specification: this consists in checking the coherence and safety of sequences of message interchanges that take place between a number of parties cooperating in carrying out some specific task. The use of type systems to formalise this kind of protocols has interested many researchers: in particular, *session types* [25,16] are recently focussed as a promising type discipline for structuring hand-shake communications. Interaction between processes is achieved by specifying corresponding sequences of messages through private channels. Such sequences are associated with session types, that assures that the two parties at each end of a channel perform consistent and complementary actions. Session types are assigned to communication channels and are shared among processes. For example, the session type `begin.?int.!bool.end` expresses that, after beginning the session, (`begin`), an integer will be received (`?int`), then a boolean value will be sent (`!bool`), and finally it is closed (`end`).

* This work was partly funded by FP6-2004-510996 Coordination Action TYPES, EPSRC GR/T03208, EPSRC GR/S55538, EPSRC GR/T04724, EPSRC GR/S68071, and EU IST–2005–015905 MOBIUS project.

M.M. Bonsangue and E.B. Johnsen (Eds.): FMOODS 2007, LNCS 4468, pp. 1–31, 2007.

Session types have been studied for several different settings, *i.e.*, for π-calculus-based formalisms [14,25,11,16,2], for CORBA [26], for functional languages [13,27], for boxed ambients [10], and recently, for CDL, a W3C standard description language for Web Services [4,28,3,24,17]. In [6] the notion of session types was investigated in the framework of object oriented languages. Such an integration has been attempted before only in [7,26] and more recently in [5].

The integration of type-safe communication patterns with object-oriented programming idioms is done in [6] via the language MOOSE, a multi-threaded object-oriented core language augmented with session types. The type system of MOOSE has been designed not only to assure the type safety of the communication protocols, but also the *progress* property, i.e. that a communication session, when started, is executed without risk that processes in a session are blocked in a deadlock state. The first property is a consequence of the *subject reduction* property, which has been shown to be a critical one for calculi involving session types. A recent article [29] analyses this issue in details, comparing different reduction rules and typing systems appeared in the literature [16,27,2,11].

The progress property, which also is an essential requirement for all kinds of applications, does not seem to have been considered before [6,7] in the literature. The operational semantics of MOOSE, however, requires communications on a channel to be synchronous, i.e. they can take place only when both processes involved in a communication are ready to perform the corresponding action. This is a strong requirement that can sometime generate deadlocks. Take for instance the parallel of the following processes:

```
1 connect c0 begin.?int.end{
2     connect c1 begin.!int.end{
3         c0.send(3);
4         c1.receive
5         }
6     }
```

Q_0

```
1 connect c0 begin.!int.end{
2     connect c1 begin.?int.end{
3         c1.send(5)
4         };
5     c0.receive
6 }
```

Q_1

Here connect c0 opens the session over channel c0, c0.send(3) sends value 3 via c0, and c0.receive receives a value via c0. These two processes in parallel, after having opened one connection on channel c0 and one on channel c1, cannot mutually exchange an integer on these channels. The resulting process would be stuck with the reduction rules of [6], since Q_0 and Q_1 are both waiting for a receiving action to synchronise.

In this paper we consider an *asynchronous* version of MOOSE, named AMOOSE: channels are buffered and can perform input and output actions at different times. This extension allows *the senders to send messages without being blocked*, reducing an overhead waiting for heavy synchronisation which the original synchronous session types require. Session types with asynchronous communication over buffered channels have been considered in [22,12] for functional languages, and in [9] for operating system services, to enforce efficient and safe message exchanges. These papers do not consider the progress property. In Java, this asynchronous semantics is found in many communication APIs such as Socket [19] and NIO [20]. Further, with the asynchrony, we naturally

(type)	t	$::= C \mid \mathsf{bool} \mid \mathsf{s} \mid (\mathsf{s},\overline{\mathsf{s}})$
(class)	*class*	$::= \mathsf{class}\ C\ \mathsf{extends}\ C\ \{\ \tilde{\mathsf{f}}\tilde{\mathsf{t}}\ \ \tilde{meth}\ \}$
(method)	*meth*	$::= \mathsf{t}\,\mathsf{m}\ (\tilde{\mathsf{t}}\,\tilde{\mathsf{x}},\tilde{\rho}\,\tilde{\mathsf{y}})\ \{\mathsf{e}\}$
(expression)	e	$::= \mathsf{x} \mid \mathsf{v} \mid \mathsf{this} \mid \mathsf{e};\mathsf{e} \mid \mathsf{e.f} := \mathsf{e} \mid \mathsf{e.f} \mid \mathsf{e.m}(\tilde{\mathsf{e}}) \mid \mathsf{new}\ C$
		$\mid \mathsf{new}\ (\mathsf{s},\overline{\mathsf{s}}) \mid \mathsf{NullExc} \mid \mathsf{spawn}\{\mathsf{e}\} \mid \mathsf{connect\ a\ s}\{\mathsf{e}\}$
		$\mid \mathsf{u.receive} \mid \mathsf{u.send(e)} \mid \mathsf{u.receiveS(x)\{e\}} \mid \mathsf{u.sendS(u)}$
		$\mid \mathsf{u.receiveIf}\{\mathsf{e}\}\{\mathsf{e}\} \mid \mathsf{u.sendIf(e)}\{\mathsf{e}\}\{\mathsf{e}\}$
		$\mid \mathsf{u.receiveWhile}\{\mathsf{e}\} \mid \mathsf{u.sendWhile(e)}\{\mathsf{e}\}$
(identifier)	a	$::= \mathsf{c} \mid \mathsf{x}$
(channel)	u	$::= \mathsf{a} \mid \mathsf{k}^+ \mid \mathsf{k}^-$
(value)	v	$::= \mathsf{c} \mid \mathsf{null} \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{o} \mid \mathsf{k}^+ \mid \mathsf{k}^-$
(thread)	P	$::= \mathsf{e} \mid P \mid P$
(heap)	h	$::= [\,] \mid h::[\mathsf{o} \mapsto (C,\tilde{\mathsf{f}}:\tilde{\mathsf{v}})] \mid h::\mathsf{c} \mid h::[\mathsf{k}^p \mapsto \tilde{\mathsf{v}}]$

Fig. 1. Syntax, where syntax occurring only at runtime appears shaded

obtain more programs with progress: in the above example, for instance, the sending actions transmit the output values to the buffered channels and running of Q_0 and Q_1 in parallel can progress and reach safely its natural end.

In [6] a single type system was defined to assure both type safety and progress. These two properties, however, are rather orthogonal: there seems to be no strong connection between them. In this paper we have chosen to define a type system for type safety and an effect system for progress. This de-coupling results in simpler systems and it allows a better understanding of the conditions needed to assure each property.

Structure of the paper. The syntax and operational semantics of AMOOSE will be introduced in Section 2, the typing system and the main definitions to formulate the subject reduction property will be introduced in Section 3. Progress properties will be discussed in Section 4. The complete proof of the subject reduction theorem is given in the Appendix.

2 Syntax and Operational Semantics

2.1 Syntax

In Fig. 1 we describe the syntax of AMOOSE, which is essentially that of the language MOOSE [6]; AMOOSE and MOOSE differ in the operational semantics, since in AMOOSE output is asynchronous, and the exchange of data between processes is realised via buffers in the queues associated to channels. We distinguish *user syntax*, *i.e.*, source level code, and *runtime syntax*, which includes null pointer exceptions, threads and heaps.

Channels. We distinguish *shared channels* and *live channels*. They both can be parameters of procedures. We deviate from [6] introducing polarised live channels [11,29]. Shared channels are only used to decide if two threads can communicate. After a connection is established the shared channel is replaced by a couple of fresh *live* channels having a different *polarity*, $+$ or $-$, one for each of the communicating threads. We denote by k^p in the same thread both the receiving channel of polarity p and the sending channel of opposite polarity $\bar{p}$: this will be clear from the operational semantics. Note that the meaning of polarities is different from that in [11], where polarities simply represent the two ends of a (unique) session channel. As a notational convention we will always use $\mathsf{c},\ldots$ to denote shared channels and $\mathsf{k}^p, \mathsf{k}_0^p, \mathsf{k}_1^p, \ldots$ to denote polarised live channels.

User syntax. The metavariable t ranges over types for expressions, ρ ranges over running session types, C ranges over class names and s ranges over shared session types. Each session type s has one corresponding *dual*, denoted $\overline{\mathsf{s}}$, which is obtained by replacing each ! (output) by ? (input) and vice versa. We introduce the full syntax of types in § 3. Class and method declarations are as usual.

The syntax of user expressions e, e' is standard but for the channel constructor $\mathsf{new}\ (\mathsf{s}, \overline{\mathsf{s}})$, which builds a fresh shared channel used to establish a private session, and the *communication expressions*, *i.e.*, $\mathsf{connect}\ \mathsf{u}\ \mathsf{s}\{\mathsf{e}\}$ and all the expressions in the last three lines.

The first line gives parameter, value, the self identifier this, sequence of expressions, assignment to fields, field access, method call, and object creation. The values are channels, null, and the literals true and false. Thread creation is declared using $\mathsf{spawn}\ \{\ \mathsf{e}\ \}$, in which the expression e is called the *thread body*. The expression $\mathsf{connect}\ \mathsf{u}\ \mathsf{s}\{\mathsf{e}\}$ starts a session: the channel u appears within the term $\{\mathsf{e}\}$ in session communications that agree with session type s. The remaining eight expressions, which realise the exchanges of data, are called *session expressions*, and start with "$\mathsf{u}.\_$"; we call u the *subject* of such expressions. In the below explanation session expressions are pairwise coupled: we say that expressions in the same pair and with the same subject are *dual* to each other.

The first pair is for exchange of values (which can be shared channels): $\mathsf{u}.\mathsf{receive}$ receives a value via u, while $\mathsf{u}.\mathsf{send}(\mathsf{e})$ evaluates e and sends the result over u. The second pair expresses live channel exchange: in $\mathsf{u}.\mathsf{receiveS}(\mathsf{x})\{\mathsf{e}\}$ the received channel will be bound to x within e, in which x is used for communications. The expression $\mathsf{u}.\mathsf{sendS}(\mathsf{u}')$ sends the channel u' over u. The third pair is for *conditional* communication: $\mathsf{u}.\mathsf{receiveIf}\{\mathsf{e}\}\{\mathsf{e}'\}$ receives a boolean value via channel u, and if it is true continues with e, otherwise with e'; the expression $\mathsf{u}.\mathsf{sendIf}(\mathsf{e})\{\mathsf{e}'\}\{\mathsf{e}''\}$ first evaluates the boolean expression e, then sends the result via channel u and if the result was true continues with e', otherwise with e''. The fourth is for *iterative* communication: the expression $\mathsf{u}.\mathsf{receiveWhile}\{\mathsf{e}\}$ receives a boolean value via channel u, and if it is true continues with e and iterates, otherwise ends; the expression $\mathsf{u}.\mathsf{sendWhile}(\mathsf{e})\{\mathsf{e}'\}$ first evaluates the boolean expression e, then it sends its result via channel u and if the result was true continues with e' and iterates, otherwise it ends.

Runtime syntax. The runtime syntax (shown shaded in Fig. 1) extends the user syntax: it adds NullExc to expressions, denoting the null pointer error; includes polarised live channels; extends values to allow for object identifiers o, which denote references to instances of classes; finally, introduces threads running in parallel. Single and multiple *threads* are ranged over by P, P'. The expression $P \mid P'$ says that P and P' are running in parallel.

Heaps, ranged over h, are built inductively using the heap composition operator '::', and contain mappings of object identifiers to instances of classes, shared channels and mappings of polarised channels to queues of values. In particular, a heap will contain the set of objects and *fresh* channels, both shared and live, that have been created since the beginning of execution. The heap produced by composing $h :: [\mathsf{o} \mapsto (C, \tilde{\mathsf{f}} : \tilde{\mathsf{v}})]$ will map o to the object $(C, \tilde{\mathsf{f}} : \tilde{\mathsf{v}})$, where C is the class name and $\tilde{\mathsf{f}} : \tilde{\mathsf{v}}$ is a representation for the vector of distinct mappings from field names to their values for this instance. The heap produced by composing $h :: \mathsf{c}$ will contain the fresh shared channel c. The heap produced by composing $h :: [\mathsf{k}^p \mapsto \tilde{\mathsf{v}}]$ will map the live channel k^p to the queue $\tilde{\mathsf{v}}$. Heap membership for object identifiers and channels is checked using standard set notation, we therefore write it as $\mathsf{o} \in h$, $\mathsf{c} \in h$, and $\mathsf{k}^p \in h$. Heap update for objects is written $h[\mathsf{o} \mapsto (C, \tilde{\mathsf{f}} : \tilde{\mathsf{v}})]$, for polarised channels $h[\mathsf{k}^p \mapsto \tilde{\mathsf{v}}]$, and field update is written $(C, \tilde{\mathsf{f}} : \tilde{\mathsf{v}})[\mathsf{f} \mapsto \mathsf{v}]$. We assume that the heap is unordered, i.e. satisfying equivalences like

$$\upsilon :: \upsilon' \equiv \upsilon' :: \upsilon, \qquad h_1 \equiv h_1', h_2 \equiv h_2' \Rightarrow h_1 :: h_2 \equiv h_1' :: h_2'$$

where υ denotes generic heap elements. With some abuse of notation the operator " :: " denotes heap concatenation without making distinction between heaps and heap elements.

2.2 Operational Semantics

This subsection presents the operational semantics of AMOOSE: the main difference with respect to [6] is that in AMOOSE output is asynchronous and the values are exchanged through queues associated to live channels in the heap. In the reduction rules then the heap plays an essential role also in communications.

We only discuss the more interesting rules. First we list the evaluation contexts.

$$\begin{array}{rl} E ::= & [-] \mid E.\mathsf{f} \mid E;\mathsf{e} \mid E.\mathsf{f} := \mathsf{e} \mid \mathsf{o}.\mathsf{f} := E \mid E.\mathsf{m}(\tilde{\mathsf{e}}) \mid \mathsf{o}.\mathsf{m}(\tilde{\mathsf{v}}, E, \tilde{\mathsf{e}}) \\ \mid & \mathsf{k}^p.\mathsf{send}(E) \mid \mathsf{k}^p.\mathsf{sendIf}(E)\{\mathsf{e}\}\{\mathsf{e}'\} \end{array}$$

Since heaps associate queues only to live channels, we can reduce only session expressions whose subjects are live channels. Moreover shared channels are sent by send, while live channels are sent by sendS. For this reason there are no evaluation contexts of the shapes $E.\mathsf{send}(\mathsf{e})$, $\mathsf{k}^p.\mathsf{sendS}(E)$ etc.

Fig. 2 defines auxiliary functions used in the operational semantics and typing rules. We assume a fixed, global class table CT, which as usual contains *Object* as topmost class.

Expressions. Fig. 3 shows the rules for execution of expressions which correspond to the sequential part of the language. The rules not involving communications are standard [18,1,8], but for the addition of a fresh shared channel to the heap (rule **NewS**$^{\rightarrow}$).