Math. Struct. in Comp. Science (2006), *vol.* 16, *pp.* 553–555.
doi:10.1017/S0960129506005305 Printed in the United Kingdom

Developments in computational models: introduction

MARIBEL FERNÁNDEZ[†] and IAN MACKIE[†‡]

[†]*King's College London, Department of Computer Science, Strand, London WC2R 2LS, U.K.*
[‡]*LIX, École Polytechnique, 91128 Palaiseau Cedex, France*

Received 15 March 2006

In recent years several new models of computation have emerged that have been inspired by the physical sciences, biology and logic, to name but a few (for example, quantum computing, chemical machines and bio-computing). Also, many developments of traditional computational models have been proposed with the aim of taking into account the new demands of computer systems users and the new capabilities of computation engines.

A new computation model, or a new feature in a traditional one, is usually reflected in a new family of programming languages, and new paradigms of software development. Thus, an understanding of the traditional and emergent models of computation facilitates the use of modern programming languages and software development tools, informs the choice of the correct language for a given application, and is essential for the design of new programming languages.

To understand what we mean by a computational model, we briefly recall a little history. The notion of computability and computable functions goes back to the beginning of the 20th century. It refers to a general notion of 'algorithm', but does not refer to a specific programming language or physical computational device. A computation model abstracts away from the material details of the programming language and the processor we are using. In the 1930s, logicians (in particular, Alan Turing and Alonzo Church) studied the meaning of computation as an abstract (mental) process, and started to design theoretical devices to model the process of computation. Since the 1930s, it has been known that certain basic problems cannot be solved by computation; the typical example being the halting problem. To prove this, Turing and Church (independently) constructed two abstract models of computation that later became the basis of the modern theory of computing: Turing Machines (designed by Alan Turing around 1936, with the aim of solving Hilbert's *decision problem*, the *Entscheidungsproblem*, which asks if there is a definite method or process by which it could be decided whether any given mathematical assertion in the functional calculus is provable), and the lambda calculus (designed by Alonzo Church, also in the late 1930s, as a foundation for the mathematical theory of functions). Kleene's theory of recursive functions was also developed in the 1930s, and gave an alternative view of computable functions. These are the 'traditional' models of

[‡] Project Logical, Pôle Commun de Recherche en informatique du plateau de Saclay, CNRS, École Polytechnique, INRIA, Université Paris-Sud.

computation; but several other models of computation, or idealised computers, have been proposed and studied since then.

It is well known that the class of computable functions is the same for all the traditional computational models, and so imperative or functional languages (which are based on the traditional models) can describe exactly the same class of *computable functions*. However, there is no equivalent thesis for some of the emergent, non-conventional, models of computation. Moreover, there is actually hope that some of these new models can solve some outstanding non-feasible problems (that is, problems that cannot be solved in a realistic time scale in traditional models). The wealth of new results and the opening of new and exciting research directions in recent years in the area of the theory of computing led us to organise the *First International Workshop on Developments in Computational Models* (DCM), in Lisbon, Portugal, on the 10th July 2005, as a satellite event of ICALP 2005, focusing on abstract models of computation and their associated programming paradigms.

The aim of DCM 2005 was to bring together researchers who are currently developing new computational models, or new features for traditional computational models, in order to foster their interaction, to provide a forum for presenting new ideas and work in progress, and to enable newcomers to learn about current activities in this area.

Thirteen papers were presented in the first DCM workshop, covering a wide range of topics: functional calculi, object-oriented languages, rewriting calculi, mobility, interaction nets, calculi for reconfiguration, quantum computing, evolutionary processors and chemical machines.

This special issue on *Developments in computational models* came out of an open call for papers after the workshop. Six articles were selected for this special issue, covering a range of computation models.

Chemical machines

The article *Generalised multisets for chemical programming* by Jean-Pierre Banâtre, Pascal Fradet and Yann Radenac provides an extension to the Gamma chemical computation model, by making the chemical reaction rules first class. This extension leads to the introduction of a higher-order chemical programming language.

Quantum computation

There are two papers addressing issues on quantum computation. First, the article *Quantum programming languages: survey and bibliography* by Simon Gay gives an introduction to quantum computing, and provides a comprehensive survey of the research to date on quantum programming languages. Simon Perdrix and Philippe Jorrand's article, *Classically-controlled quantum computation*, gives a model of quantum computation that interfaces, and is controlled by, classical computation.

The article *Reversible Combinatory Logic* by Alessandra Di Pierro, Chris Hankin and Herbert Wiklicky investigates a reversible model of combinatory logic, which was motivated by the interest in reversible computation arising from quantum computation.

Functional computations

François-Régis Sinot's article, *Call-by-need in token-passing nets*, investigates a way to bridge the gap between interaction net implementations of the λ-calculus and more traditional abstract machines, specifically with respect to reduction strategies.

Term rewriting and object calculi

The article *Addressed term rewriting systems: application to a typed object calculus* by Daniel J. Dougherty, Pierre Lescanne and Luigi Liquori introduces a new notion of rewriting on terms, which are annotated by addresses that model memory locations. The expressivity of this formalism is exploited to define a calculus of objects with explicit addresses.

Acknowledgements

We would like to thank the authors who contributed to this special issue, and the referees who worked diligently to evaluate and help improve the papers. We are grateful to all the Programme and Organising Committee members of DCM 2005 for their enthusiasm and support. Very special thanks are due to Giuseppe Longo, the editor-in-chief of *Mathematical Structures in Computer Science*, for his constant support, which has made this issue possible.