



COMPUTATION TOOLS 2025

The Sixteenth International Conference on Computational Logics, Algebras,
Programming, Tools, and Benchmarking

ISBN: 978-1-68558-264-7

April 6 - 10, 2025

Valencia, Spain

COMPUTATION TOOLS 2025 Editors

Claus-Peter Rückemann, Universität Münster / DIMF / Leibniz Universität
Hannover, Germany

COMPUTATION TOOLS 2025

Forward

The Sixteenth International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (COMPUTATION TOOLS 2025), held between April 6 – 10, 2025, continued a series of events dealing with logics, algebras, advanced computation techniques, specialized programming languages, and tools for distributed computation. Mainly, the event targeted those aspects supporting context-oriented systems, adaptive systems, service computing, patterns and content-oriented features, temporal and ubiquitous aspects, and many facets of computational benchmarking.

Similar to the previous edition, this event attracted excellent contributions and active participation from all over the world. We were very pleased to receive top quality contributions.

We take here the opportunity to warmly thank all the members of the COMPUTATION TOOLS 2025 technical program committee, as well as the numerous reviewers. The creation of quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and effort to contribute to COMPUTATION TOOLS 2025. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations and sponsors. We also gratefully thank the members of the COMPUTATION TOOLS 2025 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope COMPUTATION TOOLS 2025 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in the area of computational logics, algebras, programming, tools, and benchmarking. We also hope that Valencia provided a pleasant environment during the conference and everyone saved some time to enjoy this beautiful city.

COMPUTATION TOOLS 2025 Steering Committee

Cornel Klein, Siemens AG, Germany

Claus-Peter Rückemann, Westfälische Wilhelms-Universität Münster (WWU) / DIMF / Leibniz Universität Hannover, Germany

COMPUTATIONAL TOOLS 2025 Publicity Chairs

Francisco Javier Díaz Blasco, Universitat Politècnica de València, Spain

Ali Ahmad, Universitat Politècnica de València, Spain

José Miguel Jiménez, Universitat Politècnica de València, Spain

Sandra Viciano Tudela, Universitat Politècnica de València, Spain

COMPUTATION TOOLS 2025

Committee

COMPUTATION TOOLS 2025 Steering Committee

Cornel Klein, Siemens AG, Germany

Claus-Peter Rückemann, Westfälische Wilhelms-Universität Münster (WWU) / DIMF / Leibniz Universität Hannover, Germany

COMPUTATIONAL TOOLS 2025 Publicity Chairs

Francisco Javier Díaz Blasco, Universitat Politècnica de València, Spain

Ali Ahmad, Universitat Politècnica de València, Spain

José Miguel Jiménez, Universitat Politècnica de València, Spain

Sandra Viciano Tudela, Universitat Politècnica de València, Spain

COMPUTATION TOOLS 2025 Technical Program Committee

Lorenzo Bettini, Università di Firenze, Italy

Zineddine Bettouche, TH Deggendorf (Deggendorf Institute of Technology), Germany

Ateet Bhalla, Independent Consultant, India

Narhimene Boustia, University Saad Dahlab, Blida 1, Algeria

Azahara Camacho, RTI - Real-Time Innovations, Spain

Patricia Camacho, University of Cádiz, Spain

Angelo Ciaramella, University of Naples Parthenope, Italy

Cornel Klein, Siemens AG, Germany

Emanuele Covino, Università di Bari, Italy

Marcos Cramer, TU Dresden, Germany

Santiago Escobar, VRAIN - Universitat Politècnica de València, Spain

Andreas Fischer, Deggendorf Institute of Technology, Germany

Shengzhong Mao, University of Manchester, UK

Roderick Melnik, Wilfrid Laurier University, Canada

Corrado Mencar, Università degli Studi di Bari Aldo Moro, Italy

Ralph Müller-Pfefferkorn, Technische Universität Dresden, Germany

Keiko Nakata, SAP SE - Potsdam, Germany

Adam Naumowicz, University of Bialystok, Poland

Cecilia E. Nugraheni, Parahyangan Catholic University, Indonesia

Alberto Policriti, University of Udine, Italy

Kristin Yvonne Rozier, Iowa State University, USA

James Tan, Singapore University of Social Sciences, Singapore

Hans Tompits, Technische Universität Wien, Austria

Prajna Upadhyay, BITS Pilani Hyderabad, India

Miroslav Velez, Aries Design Automation, USA
Tao Zheng, Orange Labs China, China

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission to reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

O - A New Approach For A Very Simple Language For Distributed Computation <i>Thomas Pucklitzsch and Mathias Sporer</i>	1
---	---

O - A New Approach For A Very Simple Language For Distributed Computation

Thomas Pucklitzsch, Mathias Sporer 

Department of Computer Science

Duale Hochschule Sachsen

Glauchau, Germany

e-mail: thomas.pucklitzsch@ba-sachsen.de

Abstract—This paper describes the work on a new programming language. This language is called O and has a very simple structure. So it can be used to explain students in an easy way how a compiler works. The goal is to provide a language, that makes it as easy as possible to distribute code over the network for parallel execution and also to use different architectures to solve special problems faster.

Keywords—compiler; distributed computing; cisc

I. INTRODUCTION

There are many projects supporting the development of distributed software. The Hydro-project [1] has been working on a toolkit that can optimize the code for the needs of distributed systems. It can transform single-node software into a variety of distributed designs. The LVars project [2] uses a similar approach. LVars is a model allowing a deterministic-by-construction parallel programming. Another concept of distributed programming is the project Lasp [3]. This model was developed for large-scale applications and contains convergent data-structures as well as dataflow execution model. It provides distributed programming in large-scale applications for clients that are sometimes offline. The intention of the development of the new programming language O was to write software for a simulator called Octopus-machine. The Octopus-machine can run an individual microprogram. It comes with different ISA-Layers and different assembler languages. The simulator provides an interface for connecting to other instances and execute code on these instances. The new programming language should be configurable and include builtin functions for executing code on other nodes. These nodes could run a different ISA-Layer. This paper presents the first ideas to address these needs.

II. THE OCTOPUS MACHINE

The Octopus-machine is a simple architecture and its implementation in Python (Figure 1). Originally it was implemented for educational purpose. The architecture of the Octopus-machine based on the MIC-1 architecture which is described by Andrew Tanenbaum [4]. In difference to the Mic-1 the architecture allows to modify the microprogram easily by using the graphical user interface of the simulator software. You can adjust the ISA-Layer and add new instructions for special purposes (Figure 2). For example if you need a hardware instruction to compute the CRC-checksum you can add some microcommands and get a hw-instruction for this task. For a better usability, the simulator of the Octopus-machine includes a generator for an assembler. Following the implementation of

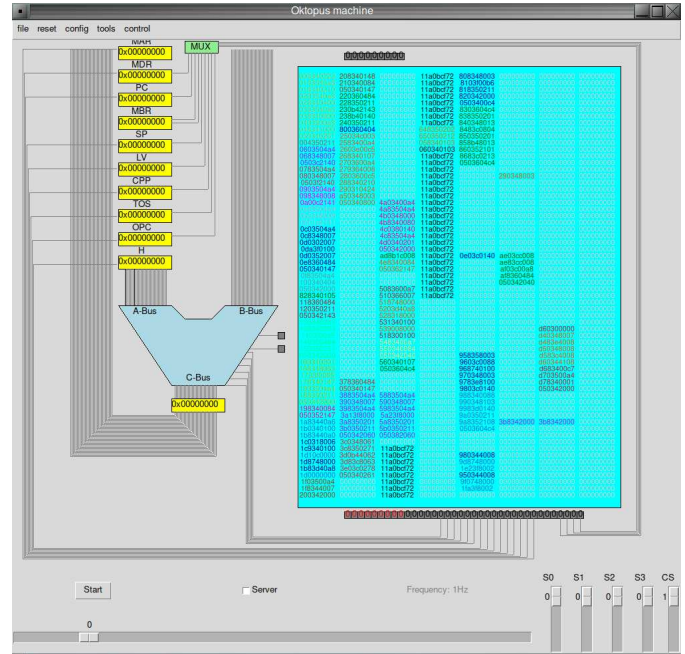


Figure 1. Oktopus software

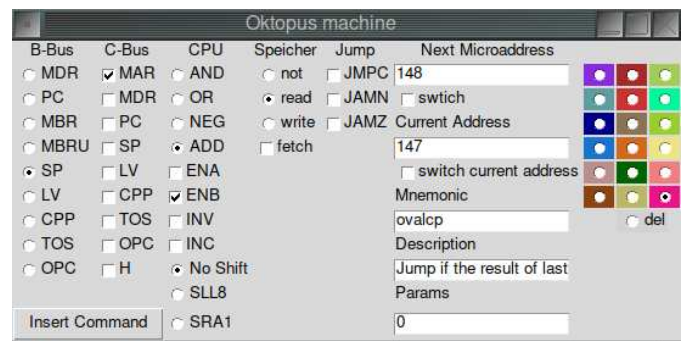


Figure 2. configuration of microcommands

your microprogram, you can generate a python script which can be used to compile the assembler program which contains the new hw-instructions. There exists also an implementation of the Octopus-machine in C for faster execution. It is possible to transfer the state of the machine and the microprogram over the network to an instance of this implementation and continue the software. So the idea was born to distribute parts of the software to different instances of the oktopus-machine. The

next step is to develop a new programming language. The programming language is referred to as O due to the name Octopus-machine.

III. HOMOICONICITY

Homoiconicity means code is data. All programming languages that support this attribute are able to generate code at runtime and execute it immediately. To send code to a instance of the Octopus-machine for execution, you can send the binary instructions to this instance and execute the code remotely. This will only work if the remote instance is running on the same Microcode. It does not work if the remote instance of the Octopus-machine is using a different microprogramm and a different ISA-Layer. In this case the solution is to transfer the source code to the remote instance. An embedded compiler inside the octopus machine must compile the source code. This means that the compiler must be very simple and the syntax of o has to be short.

IV. BUILT-IN SUPPORT FOR DISTRIBUTED EXECUTION

The implementation of the Octopus-machine includes a socket interface that allows to send data. The Idea is to use this interface, to transfer code to other instances. To enable easy distributed software development, O must provide a simple notation to specify a remote instance. To define the remote execution the symbol can be used. With `id(expression)` you can define the id of a node that should execute the expression. The tokens recognized by the parser are converted into a syntax tree by the scanner. The leaves of the syntax tree must then be replaced by sequences of machine code. Because the microprogram is individually configurable, the inserted machine code differs from machine to machine. Therefore the execution of the binarycode on another machine is only possible, if it is equipped with the same microprogram. Since this is not necessarily the case, you can write `id*(execution)`. In this case, not the compilation but rather the relevant part of the syntax tree will be transferred remotely using a binary data format. This means that the last step of the compile process can be carried out on the target node, which can then execute the resulting machine code and return the result (Figure 3).

V. THE PARSER

O keeps the parser and the scanner as simple as possible. To do this, it is defined that the length of a character string must depend on the type of token. First, the parser reads the characters from the source text and replaces all newlines with spaces. The character string is then separated using the

space character and the individual tokens are examined. The tokenizer checks each token for its length and can thus decide whether the token is an operator, a keyword or a variable. Operators consist of one character. Keywords consist of two characters and start with O and all tokens longer than two characters are recognized as variable names. The first part of the variable name indicates what data type it is. This limits the programmer and leads to keywords that are not always intuitive, but it does make it possible for the variable

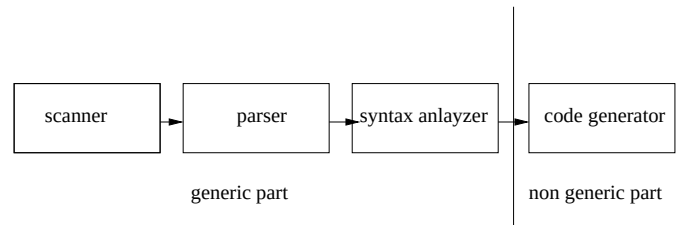


Figure 3. generic part of the compiler

types to always be recognized by their name. This ensures that errors are identified more quickly. Based on the LISP language, functions and operations are represented in the form of Polish Notation. First the operator, then the operators or the parameters of the function are specified. For example `(+ 1 2)` returns 3.

VI. CONCLUSION

The programming language O described is still a work in process. The goal is to make it as simple as possible. A lightweight compiler is needed that runs on the Octopus-machine. Due to its simplicity O is useful for educational purposes. Because of the different ISA-Layers each type of instance requires its own compiler. The compiler contains a genral part(scanner, parser) and a configurable part. O is not impemented yet. The exact definition of the language and the implementation of the parser are in process.

REFERENCES

- [1] J. Hellerstein, S. Laddad, M. Milano, C. Power, and M. Samuel, *Invited paper: Initial steps toward a compiler for distributed programs*, May 2023. DOI: 10.48550/arXiv.2305.14614.
- [2] L. Kuper and N. Ryan, *Lvars: Lattice-based data structures for deterministic parallelism*, 2013.
- [3] C. Meiklejohn and P. Van Roy, *Lasp: A language for distributed, coordination-free programming*. 2015.
- [4] A. Tanenbaum and T. Austin, *Structured computer organisation*, 2013.