When Turing meets Milner

Citation for published version (APA):

Document status and date:
Published: 01/01/2012

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 04. Oct. 2023
When Turing Meets Milner

by Jos Baeten, Bas Luttik and Paul van Tilburg

At CWI and Eindhoven University of Technology in the Netherlands, we enhanced the notion of a computation in the classical theory of computing with the notion of interaction from concurrency theory. In this way, we adapted a Turing machine as a model of computation to a Reactive Turing Machine that is an abstract model of a computer as it is used nowadays, always interacting with the user and the world.

What is a computation? This is a central question in the theory of computing, dating back to the work of Alan Turing in 1936. The classical answer is that a computation is given by a Turing machine, with the input given on its tape at the beginning, after which a deterministic sequence of steps takes place, leaving the output on the tape at the end. A computable function is a function of which the transformation of input to output can be computed by a Turing machine.

A Turing machine can serve thus as a basic model of a computation and, until the advent of the terminal in the 1970s, could also serve as a basic model of a computer. The terminal made direct interaction with the computer possible. Nowadays, a computer interacts continuously with the user at the click of a mouse or with many other computers all over the world through the Internet.

An execution of a computer is thus not just a series of steps of a computation, but also involves interaction. It cannot be modeled as a function, and is inherently nondeterministic. In our research, we made the notion of an execution precise, and compared this to the notion of a computation. To illustrate the difference between a computation and an execution, we can say that a Turing machine cannot fly an airplane, but a computer can. An automatic pilot cannot know all weather conditions en route beforehand, but can react to changing conditions run-time.

Computability theory, which is firmly grounded in automata theory and formal language theory, is the study of languages and the sets of strings, induced by them. Language can be viewed as an equivalence class of automata (under language equivalence).

The notion of interaction has been studied extensively in concurrency theory and process theory exemplified by the work of Robin Milner, who played a central role in the development of concurrency theory. He proposed a powerful parallel composition operator that is used to compose systems in parallel, including their interaction.

The semantics of concurrency theory are largely given in terms of transition systems which, in some respects, are similar to automata. There are, however, important differences. Firstly, a notion of final state, of termination, is often missing in concurrency theory. Secondly, transition systems need not be finite. The third and main difference between automata theory and concurrency theory is that language equivalence in concurrency theory is too coarse to capture a notion of interaction. Therefore, other notions of equivalence are studied in concurrency theory, capturing more of the branching structure of an automaton. Prominent among these is bisimulation equivalence.

There have been other attempts to add a notion of interaction to computability theory but none have taken full advantage of the results of concurrency theory. Studies tend not to give interaction the status it deserves; in all the formalizations of interaction machines we could find, the notion of interaction itself is still very implicit.

In our research, we used process theory to consider finite-state processes and pushdown processes. We also defined executable processes, highlighted the relationship of computable functions and executable processes, laying the foundations of executability theory alongside computability theory, and defined a new grammar for executable processes,
based on the universality of the queue process.

In conclusion, we discussed the notion of an execution that enhances a computation by taking interaction into account. This was achieved by marrying computability theory, moving up from finite automata through pushdown automata to Turing machines, with concurrency theory, not using language equivalence but divergence-preserving branching bisimilarity on automata. Although every undergraduate curriculum in computer science contains a course on automata theory, an introduction to concurrency theory is rarely included. Both theories as basic models of computation are part of the foundations of computer science, and can be taught in an integrated manner, with the results of our research [2].

Finally, it is unlikely that Turing and Milner ever did meet: by the time Milner entered King’s College in Cambridge as a young student, Turing had left some years earlier to take up a job in Manchester.

*With the help of Annette Kik, CWI*

**References:**

**Please contact:**
Jos Baeten, CWI, The Netherlands
E-mail Jos.Baeten@cwi.nl

---

**How to Compute with Metabolism in Bacteria?**

*by Claudio Angione, Pietro Liò and Giuseppe Nicosia*

An enzyme can be thought of as a computational element, i.e. a processing unit able to transform an input into an output signal. Thus, in a biochemical pathway, an enzyme reads the amount of reactants (substrates) and converts them into products. Here we consider the biochemical pathway in unicellular organisms (e.g. bacteria) as a living computer that can be programmed to obtain the desired output. Through an optimal executable code stored in the “memory” of bacteria, we can simultaneously maximize the concentration of two or more metabolites of interest.

The key role of computation in the bio-inspired science was first discovered by Alan Turing in 1952. In 1995, Bray pointed out that a single protein can transform one or multiple input signals into an output signal, and can thus be viewed as a computational or information-carrying element. Following this line of thought, we provide a framework to show that bacteria could have computational capability and act as molecular machines. Accordingly, the same framework can be applied to eukaryotic cells.

Inspired by Brent and Bruck [1], through the formalism shown in Figure 1 we describe the whole behavior of bacterial cells in terms of the von Neumann architecture. In particular, the genome sequence is thought of as an executable code specified by a set of commands in a sort of ad-hoc low-level programming language. Each combination of genes is coded as a string of L bits, each of which represents the status of a gene set. By turning off a gene set, we turn off the chemical reactions associated with it.

The memory unit contains the string y, which is the program to be executed. We model the processing unit of the bacterium as the collection of all its chemical reactions. In this way, we associate the chemical reaction network of bacteria with a Turing machine (TM). This relationship is based on the mapping between the cell metabolism and a Minsky’s register machine RM (equivalent to a TM) [3]. In fact, an RM is a multitape TM with the tapes restricted to act like simple registers (i.e. “counters”). A register is represented by a left-handed tape that can hold only positive integers by writing stacks of marks on the tape; a blank tape represents the count 0. Specifically, the reactions taking place in the cell can be thought of as increment/decrement instructions of the RM, where the RM registers (tapes) count the number of molecules of each metabolite.

Remarkably, as the biological system grows larger, reaching the desired multiple input/output, performance becomes a difficult task, necessitating some sort of machine optimization. To this end, we provide a novel algorithm called Genetic Design through Multi-objective Optimization (GDMO), with the aim of programming bacteria to maximize the yield of desired metabolites. The multi-objective optimization aims at exploiting the computational capabilities of bacteria in order to allow the maximum production of metabolites of practical or industrial interest. The solution of a multi-objective problem is a potentially infinite set of points called Pareto optimal solutions or Pareto front.

GDMO finds the genetic strategies that obey control signals, and optimizes multiple biological functions. Each point of the Pareto front provided by GDMO is a molecular machine that executes a particular task. The Pareto optimality enables not only a wide...