proceedings

Proceedings

Information Processing by Symmetric Inductive Turing Machines [†]

Mark Burgin

Department of Mathematics, University of California, 520 Portola Plaza, Los Angeles, CA 90095, USA; mburgin@math.ucla.edu

[†] Conference Morphological, Natural, Analog and Other Unconventional Forms of Computing for Cognition and Intelligence (MORCOM), Berkeley, CA, USA, 2–6 June 2019.

Published: 13 May 2020

Abstract: Traditional models of computations, such as Turing machines or partial recursive functions, perform computations of functions using a definite program controlling these computations. This approach detaches data, which are processed, and the permanent program, which controls this processing. Physical computers often process not only data but also their software (programs). To reflect this peculiarity of physical computers, symmetric models of computations and automata were introduced. In this paper, we study information processing by symmetric models, which are called symmetric inductive Turing machines and reflexive inductive Turing machines.

Keywords: information; induction; computation; symmetry; computer; efficiency

1. Introduction

Theoretical models of automata and computation have a threefold goal:

- To model physical automata and their functioning by theoretical tools;
- To study physical automata and their functioning by theoretical tools;
- To delineate the further development of physical automata and their functioning by theoretical tools.

To reflect important properties of computers, Marcin Schroeder introduced a new theoretical model of computation—symmetric Turing machines or S-machines [1,2]. In a conventional Turing machine, the head (processor) performs operations with data in the memory (tape) using a fixed system of instructions—its program. In a symmetric Turing machine, information processing goes not only from the head to the memory but also backward. On the one hand, the head (processor) performs operations with data in the memory using a fixed system of instructions—its program. On the other hand, the memory performs operations with instructions from the head (processor).

It is also possible to carry out this approach using two types of memory—data memory and program memory—and having a processor that performs two kinds of operations—operations with data based on information stored in the program memory and operations with the program based on information stored in the data memory.

As we know, physical computers do not only process data but also perform operations with their programs using special tools, such as interpreters, compilers, and translators. There are also program optimizers, which improve characteristics of programs transforming these programs.

Automata that perform transformations with their programs, such as reflexive Turing machines, were explored in [3]. It was proved that these machines have the same computing power as Turing machines. This result disproved Kleene's conjecture [4], which suggested that algorithms that change their programs while computing would be more powerful than Turing machines.

However, it was also proved that reflexive Turing machines could be much more efficient than conventional Turing machines [3].

Using a technique similar to the one employed in [3], it is possible to prove that in a general case, functioning of a symmetric Turing machine working in parallel mode can be simulated by a conventional Turing machine with five tapes and five heads. It means that symmetric Turing machines have the same computing power as Turing machines.

To achieve higher computing power, here we introduce and study symmetric inductive Turing machines, which further develop the structure and possibilities of inductive Turing machines allowing modeling of natural computations in various situations.

2. Symmetry in Computations

Physical computers and networks process information using various programs. The system of these programs is called the software of a computer or network, of which the processed information is called infware [5].

There is a definite symmetry between software and infware. To explicate this symmetry, we need a broader understanding of a program. Namely, we define a *program of computation* as a structure that determines (controls) a computational process.

A computational process transforms, transmits, and stores structures, which form infware of the computing system. Usually these structures represent data, although the goal of computing devices is working with knowledge.

Thus, we have two types of structures—controlling and processed structures. However, the roles of these two types of structures can be exchanged—computers can process their programs, e.g., translate them from one programming language to another or optimize them, using data structures for controlling this processing.

This situation displays symmetry between software and infware. Utilization of this symmetry can serve for better organization and efficient optimization of computational processes.

When a computational process involves transformation of not only infware but also software, it becomes symmetric. Consequently, automata and machines that process their infware and software are called *symmetric*.

Note that automata and machines can process their own software or software of other automata and machines. In the first case, corresponding automata and machines are called *reflexive*. In the second case, corresponding automata and machines are called *exterior*. For instance, translators and interpreters are exterior symmetric programs. Automata and machines that can process both their own software and software of other automata and machines are called *coalescent*.

These peculiarities are reflected in some directions in computer programming, such as reflective programming or metaprogramming. They allow computer programs to be observing, reading, generating, analyzing, and modifying other computer programs or themselves at runtime [6–9]. The goal is to allow software developers to minimize the length of code to express a solution, in turn reducing the development time. Reflective programming and metaprogramming also provide means for higher flexibility of programs to efficiently handle new situations without recompilation.

Metaprogramming and reflective programming techniques were popular in the 1970s and 1980s when they were based on list processing languages such as LISP. In particular, LISP hardware machines were employed in the 1980s enabling applications that could process code and were useful for artificial intelligence applications.

There are three modes of temporal organization of symmetric information processing:

– *Program preprocessing,* when at first, the program of the S-machine is processed and then it is used for input data processing;
– *Interchangeable processing,* when the whole computational process is divided into intervals such that an interval of data processing is followed by an interval of program processing, which in turn is followed by an interval of data processing, and so on;
– *Concurrent processing,* when data processing and program processing are performed at the same time.

The latter mode is used in reflexive Turing machines [3]. To organize concurrent processing, a reflexive Turing machine uses two copies of the same program—the execution copy and the processed copy. One is used for data processing, while the second one is transformed, for example, optimized, then at some step of computation, a copy of the processed copy changes the execution copy and is used for data transformation, while the machine continues to transform the processed copy.

3. Reflexive Inductive Turing Machines

Thus, we have three classes of symmetric automata and machines:

→ *Reflexive symmetric automata and machines*, which process their own software in addition to their infware;

→ *Exterior symmetric automata and machines*, which process software of other automata and machines in addition to their own infware;

→ *Coalescent symmetric automata and machines*, which process software of other automata and machines in addition to their own software and infware.

Here, we are mostly interested in reflexive symmetric automata and machines, describing how to upgrade inductive Turing machines, enhancing them with symmetric computations.

We remind a definition of an inductive Turing machine [5].

The *hardware* of an inductive Turing machine *M* consists of three abstract devices:

- The *control device* (*controller*) *A* is a finite automaton, which controls functioning of *M;*
- The *operating device* (*processor*) *H* can include any number of processors;
- The memory *E*.

The *software* of an inductive Turing machine *M* consists of rules according to which the operating device *H* functions.

The *infware* of an inductive Turing machine *M* consists of those symbols, words, and languages, which are processed by *M*.

The memory *E* of an inductive Turing machine of the first and higher orders is a network of cells structured by a system of relations that provide connections between cells. This structuring determines input registers, the working memory, and output registers of *M*. Cells can be of different types: *binary cells* store bits of information represented by symbols 1 and 0; *byte cells* store information represented by strings of eight binary digits; *symbol cells* store symbols of the alphabet(s) of the machine *M* and so on.

The *operating device* (*processor*) *H* of an inductive Turing machine consists of one or several heads, similar to those which are used in Turing machines.

An inductive Turing machine gives the result either when it comes to a final state or when the word in the output register stops changing and this word becomes the result of computation. This is the *output-stabilizing mode* of computation. There are also other ways to determine results of inductive computations.

A reflexive inductive Turing machine utilizes two types of memory—data memory and program memory. Namely, to enhance an inductive Turing machine with the possibility to perform symmetric computations, we separate the memory *E* into two parts—E_d, which is used for working with the infware (e.g., data) and E_d, which is used for working with the software (i.e., instructions) of the machine *M*.

In addition, a reflexive inductive Turing machine has two processors—a data processor and a program (instruction) processor. The data processor performs operations with data based on information in the form of a program stored in the program memory. Similarly, the program (instruction) processor performs operations with the program based on information stored in the data memory. This information is obtained, interpreting definite data as a program.

Both processors function under the control of the same control device *C*, which organizes their interaction, which can be:

- *Separated*, implying program preprocessing when at first, the program of the S-machine is processed and then it is used for input data processing;
- *Interchangeable*, when the whole computational process is divided into intervals such that an interval of data processing is followed by an interval of program processing, which in turn is followed by an interval of data processing, and so on;
- *Parallel*, when processing of programs go parallel with processing data under the regulation of the same control device.

Note that utilization of the common control device for data and program processors does allow the achievement of true concurrency, which demands separate control devices.

Using techniques similar to ones elaborated in [3], it is possible to prove the following results.

Theorem 1. *Functioning of a reflexive inductive Turing machine can be simulated by an inductive Turing machine of the same order*.

Theorem 2. *For any natural number k, any universal inductive Turing machine U of the first order and any program p for U with time complexity* $T_{U,p}(n) \geq n$*, there is a reflexive inductive Turing machine M of the first order that can perform the same computations with time complexity* $T_{M,p}(n)$*, which is asymptotically less than* $(½)^{-k}\, T_{U,p}(n)$.

These results show that although reflexive inductive Turing machines have the same computing power as inductive Turing machines, they can compute much faster.

4. Conclusions

Symmetric inductive Turing machines were defined and their special case—a reflexive inductive Turing machine—was studied. It was demonstrated that introduction of symmetric computations allows a decreasing time of computation although it does not change the computing power of the machines working with one control device.

It was proved that reflexive inductive Turing machines have the same computing power as inductive Turing machines when they process data and programs under the regulation of the same control device. Utilization of separate control devices for data and program processing allows the achievement of true concurrency, which, in a general case, can bring forth computation of recursively incomputable data. Thus, there is an open problem in whether symmetric inductive Turing machines in general, and reflexive inductive Turing machines in particular, can be more computationally powerful than inductive Turing machines.

An interesting problem for future research is elaboration of algorithmic (Kolmogorov) complexity based on symmetric Turing machines and symmetric inductive Turing machines. It has been proved [5] that inductive Turing machines can essentially decrease algorithmic (Kolmogorov) complexity for infinitely many constructive objects in comparison with Turing machines. It would be interesting to find whether symmetric inductive Turing machines can essentially decrease algorithmic (Kolmogorov) complexity for infinitely many constructive objects in comparison with symmetric Turing machines.

Funding: This research received no external funding

Conflicts of Interest: The author declares no conflict of interest.

References

1. Schroeder, M.J. Dualism of Selective and Structural Manifestations of Information in Modelling of Information Dynamics. In *Computing Nature*; SAPERE 7; Springer: Berlin, Germany, 2013; pp. 125–137.
2. Schroeder, M.J. From Proactive to Interactive Theory of Computation. In Proceedings of the 6th AISB Symposium on Computing and Philosophy: The Scandal of Computation—What is Computation? Exeter, UK, 2–5 April 2013; pp. 47–51.

3. Burgin, M. Reflexive Calculi and Logic of Expert Systems. In *Creative Processes Modeling by Means of Knowledge Bases*; Institute of Mathematics: Sofia, Bulgaria, 1992; pp. 139–160. (In Russian)
4. Kleene, S. Constructive and Non-Constructive Operations. In Proceedings of the International Congress of Mathematicians, Edinburg, UK, 14–21 August 1958; Cambridge University Press: Cambridge, UK, 1960.
5. Burgin, M. *Superrecursive Algorithms*; Springer: New York, NY, USA, 2005.
6. Smith, B.C. Procedural Reflection in Programming Languages. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1982.
7. Demers, F.-N.; Malenfant, J. Reflection in logic, functional and object-oriented programming: a Short Comparative Study. In Proceedings of the IJCAI'95 Workshop on Reflection and Metalevel Architectures and their Applications in AI, Montreal, QC, Canada, 20–25 August 1995; pp. 29–38.
8. Chlipala, A. Ur: statically-typed metaprogramming with type-level record computation. In Proceedings of the ACM SIGPLAN 2010 Conference on Programming Language Design and Implementation (PLDI'10), Phoenix, AZ, USA, 22–28 June 2010; Volume 45, pp. 122–133.
9. Ibrahim, M.H. Reflection in Object-Oriented Programming. In Proceedings of the IJCAI'95 Workshop on Reflection and Metalevel Architectures and their Applications in AI, Montreal, QC, Canada, 20–25 August 1995; pp. 29–38.