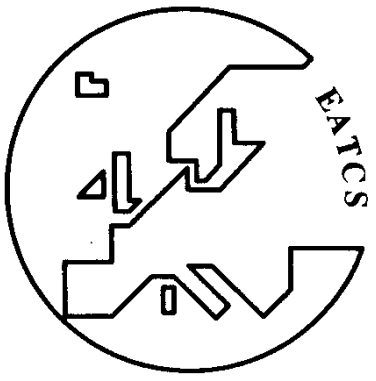
ISSN 0252–9742

Bulletin

of the

European Association for Theoretical Computer Science

EATCS



Number 81 October 2003

§3

Epilogue

We extended the concept of term (or syntactic) unification for allowing the search of unknown functions, remaining in the framework of first order logic. Unifiability is now equivalent to consistence. The unification algorithm should be a modification of the completion algorithm for enumerating a confluent system that allows cancellation of ground symbols. A "confluence criterium", like the one with critical pairs, should indicate what equations are to be added to warranty confluence and could help to eventually stop enumeration. For example, orientations of $dec(\{s = t\})$ for critical pairs (s, t) of all possible instances of equations could be added, but this is not practicable. The restriction to critical pairs taking most general unifiers of non-variable subterms of left sides of equations leads to a confluent system if the result is a terminating TRS, but in the general case it is not necessarily confluent. Unfortunately, known confluence criteria without demanding termination are complicated and demand strong restrictions to the TRS. The practical use of this extended concept of unification depends on future research for finding relevant families of equation systems having a practicable algorithm.

References

[1] Stephen Cole Kleene, Recursive Predicates and Quantifiers, Transactions of the American Mathematical Society, V 53, 1943.

[2] Stephen Cole Kleene, "Introduction to Metamathematics", Wolters-Noordhoff Publishing - Groningen, North-Holland Publishing Company - Amsterdam, First Published 1952, Sixth Reprint 1971.

[3] Jan Willem Klop, Term Rewriting Systems, In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors: Handbook of Logik in Computer Science, volume 2, chapter 1, pages 1–116. Oxford University Press, 1992.

[4] A. Martelli und U. Montanari, "An Efficient Unification Algorithm", ACM Transactions on Programming Languages and Systems 4, 2 (April 1982), pages 258-282.

[5] Dag Prawitz, "Natural Deduction", Almqvist & Wiksells, Uppsala, 1965.

[6] Readi Nasser, R. mj-Prolog, 1. Proof Theoretical Foundations, Internal Report, CIS, Uni-München, ftp://ftp.cis.uni-muenchen.de/pub/cis-berichte/CIS-Bericht-99-124.ps (Note lapse in lines 12–13 of §32).

[7] A. S. Troelstra, H. Schwichtenberg, Basic Proof Theory, Cambridge University Press, 1996.

goodgurvey & refs

BEYOND TURING MACHINES

Eugene Eberbach
Comp. and Inf. Science Dept.
University of Massachusetts
North Dartmouth, MA 02747
eeberbach@umassd.edu

Peter Wegner
Dept. of Computer Science
Brown University
Providence, RI 02912
pw@cs.brown.edu

Abstract

In this paper we describe and analyze models of problem solving and computation going beyond Turing Machines. Three principles of extending the Turing Machine's expressiveness are identified, namely, by *interaction, evolution* and *infinity*. Several models utilizing the above principles are presented. Other models, instead of directly attacking undecidability, concentrate rather on intractable problems. Four ways to deal with intractability have been derived. These are *interaction, evolution, guessing* and *parallelism* principles. Both types of models, dealing with Turing Machine undecidability or intractability, are expected to better answer the needs of new computer science in 2000s.

1 Introduction

Alan Turing was born in 1912, became an undergraduate in mathematics at King's College, Cambridge in 1931, and published a paper in 1936 whose model of Turing Machines became a central basis for the theory of computing. This paper, entitled "On Computable Numbers with an Application to the Entscheidungsproblem" [34] was primarily a mathematical demonstration that computers could not model mathematical problem solving.

The *Entscheidungsproblem* had been adopted by Hilbert as a basis that all mathematical problems could be proved true or false by logic. Hilbert's thesis had been accepted by many mathematicians, including Russell and Whitehead in their book "Principia Mathematica" [42], but was disproved by Gödel in 1931 [18]. Turing's disproof of Hilbert's *Entscheidungsproblem* was accepted by Gödel as better than his own proof, and was also accepted by Church as better than his own attempt to show that the λ-calculus could not prove the *Entscheidungsproblem* [8]. Church invited Turing to Princeton for further discussion of Turing Machines as a

method of solving mathematical and computational problems, and their work led to the acceptance of the "Church-Turing Thesis" as a uniform model for computing.

In the 1960s, with the development of new ACM computer science curriculum based on algorithms, Turing machines were chosen to fulfill a central role in computer science, because of its elegance and simplicity, and for many years they have been useful in that function. However, we forget typically that Turing was more concerned that the *Entscheidungsproblem* is not solvable, and mathematics cannot be fully described by algorithms, rather than to propose or create the foundations of computer science. Turing himself did not accept Turing Machine as a complete model for problem solving, and its acceptance by theoreticians contradicted Turing's view on this subject.

In particular, Turing besides his classical a-machines (automatic machines, known commonly as TMs), proposed more expressive TMs with oracles (o- machines) [35], choice machines (c-machines) [34], and unorganized machines (u-machines) [36]. Ulam and von Neumann in the search of a universal constructor (subsuming universal computability) used the help of expressive power of cellular automata rather than TMs. Several scientists were attracted by expressiveness of neural networks, analog computers, automata networks. Wegner extended the classical TM by adding interaction in his Interaction Machines as a form of open systems. Closely related to that is research on process algebras standardizing interaction by various forms of message passing. For example, Milner in his Turing award lecture [29] pointed out that his π-calculus is a model of interactive systems requiring extension of the λ-calculus (and as a consequence TMs) approach to problem solving because it does not capture interactive concurrent computing. Hoare's CSP [22] and Eberbach's $-calculus [11] are additional examples of interactive models. Van Leeuwen and Wiedermann use interactive Turing machines with advice (oracle) as a basic building block of their Site and Internet machines [43].

In the next section we discuss why Turing Machines are not complete to describe all possible computations. We introduce the notions of superTuring computation and superTuring computers. In section 3, we present three methods leading to more expressive models than Turing machines. They are called: interaction, evolution and infinity principles. Based on those results, in section 4 we present several models going beyond Turing machines. In section 5, we describe four ways to approach intractability. Section 6 overviews several models dealing with hard computational problems. Section 7 contains conclusions.

2 SuperTuring Models of Computation

After 60 years of unsuccessful attempts to find a stronger model of computation than TM, most of computer scientists gave up and silently accepted Church-Turing Thesis as describing all possible computation. There is no problem with equating classical algorithms and Turing Machines. The problem is with equating all forms of computation (i.e., an extension of the Church-Turing thesis to, so called *strong Church-Turing thesis*) with algorithms and Turing Machines. Algorithms are elegant and useful, and it would be nice if it was possible to describe all computations by algorithms. However, this is not possible. There are several areas which cannot be described properly using algorithms and Turing Machines.

For example, the TM model is too weak to describe properly the Internet, evolution, or robotics because it is a closed model, which requires that all inputs are given in advance, and TM is allowed to use an unbounded but only finite amount of time or memory resources [12, 14, 40, 41].

In the case of the Internet, the web clients "jump" into the middle of interaction, without a knowledge of the server state, and previous history of other clients. A dynamic set of inputs and outputs, parallel interaction of multiple clients and servers, a dynamic structure of the Internet communication links and nodes, is outside what a sequential, static and requiring full specification Turing Machine can represent.

TMs have also problems with capturing evolution, because solutions and evolutionary algorithms are changed in each generation, and the search for a solution (represented by a global optimum) is in a general an infinite process. The above is true independently whether search spaces are finite or infinite. This is so because the search for a solution in evolutionary computation is typically probabilistic, and then the infinite number of steps may be required for finite search spaces as well.

Robots interact with environments which are very often more complex than robots themselves, and even worse - the environments can be noncomputable in the sense of the TM. Thus it is clear why conventional computer science - GOFAI (Good Old Fashioned AI) failed miserably, when tried to build a model of a robot and its environment using Turing Machines and algorithms (i.e., deliberative goal-based robots building the model of the world to plan their actions, versus behavior-based robots reacting only to their sensory inputs, because algorithmic planning did not work).

The biggest challenge seems to be the development and acceptance of computational models more expressive than Turing Machines, referred here as superTuring models of computation.

- By *superTuring computation* we mean any computation that cannot be carried out by a Turing Machine as well as any (algorithmic) computation car-

ried out by a Turing Machine.

- We define a *superTuring computer* as any system or device capable of carrying out superTuring computation.

In this paper, we will use the term computation in this wider "superTuring computation" sense, and not in the narrower "algorithmic" meaning.

3 Three Ways to Extend the TM Expressiveness to SuperTuring Models of Computation: The Interaction, Evolution and Infinity Principles

Note that TMs are supposed to solve any problem in a finite number of steps (discrete time), each step should be well defined, The TM sees only a finite amount of discrete information (a symbol - contents of the cell - under read/write head) at the time, The TM gets all its input from the passive environment in advance at the beginning of computation, and outputs the result of a solved problem to the environment at the end of computation.

On the other hand, real computers interact with each other and an environment perhaps many times during the computation, and produce possibly not a single output, but an infinite sequence of results. We can guess that if we allow the environment to help/participate in computation, and we release restrictions on boundedness of time, memory, or nondeterminism, then the resulting "Turing machine" may be more expressive, i.e. able to solve a larger class of problems than the classical TM.

Yes, if we allow the TM to get a help from a smarter outside component (call it an agent, environment, or oracle), or the TM will evolve to a smarter version of itself, or the TM will have an access to infinite resources (e.g., time or memory), then we can conjecture that such machine might be more powerful.

This is consistent what other researchers have observed, and what we extracted from several models extending TM and what will be in more details discussed in section 4. For example, in the search for more expressive than TM models, van Leeuwen and Wiedermann [43] stressed, so-called non-uniformity of programs (upgrade of hardware and system software of computers, referred here as evolution) , interaction of machines (referred here as interaction), and infinity of operators (referred here as infinity) of modern computing leading to models more expressive than Turing machines.

It is at least possible to extend classical Turing machines in three different ways:

1. **Interaction**: opening a closed model by *interacting* with the external world. The world can be in the form of a singular entity or multiple agents, which actively participate in computation. The external component has to be either "smarter" than TM or it has to be infinite many of "dumb/ordinary" components. This approach is represented by Turing c-machines and o-machines, Wegner's Interaction Machines [37], and by Van Leeuwen's Site and Internet machines [43].

   Interactive computation involves interaction with an external world, or the *environment* of the computation, *during* the computation – rather than *before* and *after* it, as in algorithmic computation.

   When an interactive system consists of many autonomous (or asynchronous) concurrent components, it cannot in general be simulated by interleaving the behaviors of its subsystems. *Distributed interaction is more expressive than sequential interaction, just as sequential interaction is more expressive than algorithmic computation.* Multi-component systems are capable of richer behaviors than sequential agents. Their behaviors are known as *emergent*, since they emerge as the property of the whole system without being present, in whole or part, in any of its components.

2. **Evolution:** allowing adaptation of TM to a possibly smarter component or an infinite sequence of components. The Turing Machine stops being static but continuously evolves, so it is able to solve new types of problems. perhaps impossible to solve by its previous incarnation. Turing's unorganized machine learning [36] can be viewed as an example of this. He proposed strategies, now known as *genetic algorithms*, to evolve the connections between the neurons within the u-machine.

   In general, evolution can be done by upgrade of either hardware or software, by self-adaptive, learning programs, or evolvable, self-reproductive hardware. Evolution may happen in continuous or discrete steps, leading possibly to capabilities previously not present. In particular, an ordinary Turing Machine can evolve to one with an oracle, or to a persistent TM that does not reinitialize its tape before computations, or one that replicates its tapes indefinitely, or to an infinite network of cells, each being a separate instance of TM or Finite State Machine. The possibilities of evolution (the types of variation operators) are endless.

   Evolution can be controlled by interaction with the environment, or by some performance measure, such as its *fitness*, or *utility*, or *cost*. The evolution principle is used by *site and internet machines* [43], *$-calculus* [11], and Turing's *u-machines* [36].

3. **Infinity:** releasing restriction on boundedness of resources, i.e., allowing either an infinite initial configuration, infinite many computing elements, infinite time, or infinite/uncountable alphabets. For example:

   - *Allowing an infinite initial configuration (persistence)*: is represented by cellular automata, Interaction Machines [37], Persistent Turing Machines [20] and $-calculus [11].
   - *Allowing infinite many computing elements (infinity of operators)* can be modeled by an infinite number of tapes of TM, or infinite number of read/write heads - an unbounded parallelism. The approach is represented by cellular automata [44], neural networks, random automata networks [17], π-calculus [28], $-calculus [11]. It should be used by models of massively parallel computers or Internet, where we do not put restriction on the number of computing elements. Note that we should allow an infinite number of computers modeling massively parallel computers and Internet, because by the same argument as Turing allowed (non-existing in nature) an infinite tape in TM, we should allow an infinite supply of (non-existing in nature) machines in models of parallel and distributed computation.
   - *Allowing infinite time* is represented by reactive systems.
   - *Allowing uncountable alphabets* is represented by analog computers, neural networks and hybrid automata [32].

Each of these extensions separately leads already to models more expressive than Turing machines, i.e. it is sufficient only to fulfill exactly one of (1), (2), or (3) to be outside of TM model. Note that the approach to extend TM in (3) is different from (1) and (2). In (1) and (2), we assume that TM may evolve to or get help from a smarter computational element, which may not have a well defined semantics/structure (e.g., in the form of oracle producing non-recursive output. However, we do not have a look inside of the internal structure of such powerful component, besides that it somehow does what TM was unable to do on its own). This is equivalent to release the restriction that each step of algorithm has to have a well defined semantics (however, as was observed by Turing and his followers, it would be impractical to construct commercial computers based on magic of oracles, thus only Turing machines were called automatic machines). On the other hand, in (3) we manipulate with the restriction on boundedness of resources, which some people may claim to be unrealistic as well (or belonging to miracles too). However, such an approach is quite common in mathematics and in real world, when we are not interested in restriction on available resources. If somebody still feels uneasiness with such an approach, please note that TMs are

allowed to use unbounded tapes in their computations, and cellular automata - to employ infinite number of cells.

Whether a list of three possible TM extensions is complete, it is not clear. At this point, we are rather interested that such extensions are possible, and that they cover all models discussed in section 4.

4 Beyond TM Undecidability: The SuperTuring Models of Computation

In this section, we will present several models more expressive than TMs. We will also point out which method, i.e., interaction, evolution, or infinity was used to gain a higher expressiveness of the specific model.

4.1 Choice Machines and Oracle Machines

In his 1936 paper [34], besides a-machines, Turing also proposed "choice" machines (c-machines) as an alternative model of computation. In contrast to the closed-world approach of a-machines, c-machines interacted with an operator during the computation. Unlike a-machines, a c-machine's "motion is only partially determined by the configuration"; in certain configurations, it stops and "cannot go until some arbitrary choice has been made by an external operator".

In 1939 Turing introduced "oracle" machines (o-machines) [35]. In ancient Greece, "oracles" were people who others consulted for advice. Similarly, Turing's oracle can be queried about any value. An oracle is formally viewed as a set. It will return *true* if the queried value is in this set, and *false* otherwise. *Oracles* in ancient Greece were believed to possess access to hidden knowledge that came to them directly from the deities. Analogously, Turing's oracles are allowed to represent *non-computable* information. Turing had the following explanation how that is possible: "we shall not go any further into the nature of this oracle apart from saying that it cannot be a machine". Thus Turing's oracle represents a non-decomposable "black box", enabling an a-machine to solve TM unsolvable problems.

Both c-machines and o-machines gain higher expressiveness by an *interaction* principle, In c-machines and o-machines, a-machines interact with a smarter external component (oracle, or the external operator, respectively). It is possible (but not necessary) that the set of oracle's values or the number of choices for the external operator is infinite, thus the *infinity* is not required for o-machines or c-machines to be more expressive than TM.

4.2 Cellular Automata

Cellular automata [44, 6], have been introduced by John von Neumann on suggestion of Stan Ulam to study self-reproduction, universal computability and universal constructability. Cellular automata have held a lasting appeal through decades for a number of reasons. Notable among them is the simplicity of their finitary specification. The richness of their behavior (they can easily simulate Turing machines and solve problems unsolvable by Turing machines) seems to fit more tightly with our intuition about physical space. Their homogeneity bears promise of a simple hardware (robot) implementation.

Formally, a cellular automaton [17] is a pair $(D, \{M\})$ consisting of a cellular space $(D, \{Q\})$ (countably infinite, locally-finite regular (each node has the same degree) directed graph D with states Q assigned to each node) and common finite-state machine M with input alphabet $\Sigma = Q^d$, $Q^d = Q \times ... \times Q$ (d times) and local transition functions $\delta : Q \times \Sigma \rightarrow Q$.

The global evolution (dynamics) of a cellular automaton is best viewed as a discrete dynamical system (self-map) $T : C \times C$, where $C = \Pi_i Q$ is a set of configurations (total states) $T(x)_i = \delta(x_i, x_{i_1}, ..., x_{i_d})$. A cellular automaton operates locally as follows. A copy of a common finite state machine M occupies each vertex of a regular graph (cellular space) which is then called a cell. Synchronously, each copy of M looks up its input in the states $x_1, ..., x_d$ of its neighboring cells and its own state x_0, and then changes its state according to its local dynamics δ. The cellular automaton performs its calculation by repeating these atomic local rules any (possibly very large) number of times for all sites resulting in emergent behavior (global dynamics) of the whole system.

Turing machines can be simulated by 1D cellular automata but there exist problems solvable by CA, but not by TM (for example, 1D CA taking as an input a real number, represented as an infinite binary expansion, and stabilizing iff it is an integer. A TM cannot solve this problem since the real number can be given as 0.999..., and hence TM will not even finish reading its input in finite time). Thus we obtain that $TM \subset CA$.

The expressive power of cellular automata is due to an unbounded number of cells - the *infinity* principle. Note that each cell is described by finitary means only, and forms a very restricted form of TM - a finite-state machine. The expressiveness of cellular automata is due to interaction of infinite number of cells.

4.3 Unorganized Machines

It is interesting that Turing himself considered models resembling cellular automata and neural networks. Neural Networks have their roots in forgotten Turing's u-machines (1948) [36] Pitts-McCulloch neurons (1943) [26], Rosenblatt's

perceptron (1958) [31].

In the work which remained unpublished until 1968 (this is probably the main reason that it is omitted in practically all textbooks on neural networks), Turing introduced a kind of neural network that he called a "B-type unorganized machine" *u-machines*, which consists of artificial neurons and devices that modify the connections between them. B-type machines may contain any number of neurons connected in any pattern but always subject to restriction that each neuron-to-neuron connection must pass through a modifier. Turing showed that a sufficiently large B-type neural network can be configured (via its connection modifiers) in such a way that it becomes a general-purpose computer. It is unclear, to which extend Pitts-McCulloch work influenced Turing in his work on u-machines.

If we allow an infinite supply of neurons in u-machine, then u-machine by the *infinity principle* will be a more expressive than TM.

4.4 Interaction Machines

Interaction Machines have been introduced by Wegner in 1997 to describe interaction of object-oriented and distributed systems [37, 38, 39].

The paradigm shift from Algorithms to Interaction captures the technology shift from mainframes to workstations and networks, from number crunching to embedded systems and user interfaces, and from procedure-oriented to object-based and distributed programming. In [37] it is stated that Turing Machines are too weak to express interaction of object-oriented and distributed systems, and Interaction Machines (IMs) are proposed as a stronger model that better captures computational behavior for finite interactive computing agents. The intuition that computing corresponds to formal computability by Turing machines breaks down when the notion of what is computable is broadened to include interaction. Though Church's thesis is valid in the narrow sense that Turing machines express the behavior of algorithms, the broader assertion that algorithms precisely capture what can be computed is invalid [38].

Interaction machines extend the Turing machine model by adding *interaction*, i.e., input and output actions (read and write statements) opening the closed world of Turing machines. Whereas Turing machines require all inputs appear on the tape prior to the computation and shut out the world during the process of computation, Interaction Machines allow inputs to be dynamically generated and require inputs to be represented by a potentially infinite stream, since any finite stream can be dynamically extended by adversaries. Interactive systems interact with an external environment they cannot control. Adversaries have the last word and can always extend any finite sequence. The behavior of adversaries is better modeled by infinite processes that express the cardinality of the real numbers (Cantor diagonalization) than by enumerable sequences. Interaction Machines (IM) cor-

respond to Turing machines with unbounded tape contents. Whereas the number of inputs of a Turing machine are countable, the number of potential streams of an interaction machine is nonenumerable. However, IMs, dissimilar to TMs, use *interaction* as a basic principle of their work which is a foreign notion for closed world of TMs. TM "knows" in advance the contents of its tape, IM never is sure which input will receive as the result of interaction. The evolution (or dynamicity) of bounded inputs and outputs differs IMs from TMs.

Interaction Machines consist of Sequential Interaction Machines (SIMs) and Multi-Stream Interaction Machines (MIMs). SIMs are stream processing machines that model sequential interaction by I/O streams. *Sequential Interaction Machines (SIMs)* are state-transition machines

$$M = (S, I, m), \textit{where}$$

S is an enumerable set of states, I is an enumerable set of dynamically bound inputs, and the transition mapping $m : S \times I \rightarrow S \times O$ maps state-action pairs into new states and outputs. Each computation step $(s, i) \rightarrow (s', o)$ of a SIM can be viewed as a complete TM computation, where i is dynamically supplied input token (string), the output o may affect subsequent inputs, and s' is the next state of the SIM. Elements of both S and I are finite at any given time but their size is unbounded.

Multi-stream Interaction Machines (MIMs) are finite agents that interact with multiple autonomous streams: for example, distributed databases or ATM systems that provide services to multiple autonomous clients.

The expressiveness of Interaction Machines is due to all three principles: *infinity*, *evolution* and *interaction*, i.e., unbounded tape contents, enumerable (potentially infinite) set of states, inputs, and outputs. The set of inputs is dynamically bounded (it evolves). Additionally, it is assumed that other agents (environment to interact with) can be non-computable, which by itself leads to more expressive model than TM.

4.5 Persistent Turing Machines

Persistent Turing Machines (PTMs) are a stream-based extension of the Turing machine model by adding interaction and persistency [20, 21]. They were introduced as a minimal extension to the classical Turing-machine model (persistence of the work tape) needed for expressiveness going beyond Turing machines. They can also be thought as a canonical form for Sequential Interaction Machines [39].

PTMs correspond to non-deterministic 3-tape Turing machines, each equipped with an input, work, and output tape. However, not 3 tapes make PTMs more expressive than TMs (it is a well known that nondeterministic Turing machines with

a finite number of tapes can be simulated by a single-tape deterministic TM). Three tapes allow in a more elegant way to separate technically input, output and scratch streams in a similar way as RAM and cache memory is separated from ports and input/ouput buffers in modern computers. Exactly, the persistence of the contents of the "scratch"/work tape makes PTMs more expressive than "amnesic" ordinary Turing machines. This is consistent with results from other areas that adding memory matters, and systems with some kind of persistence are more expressive compared to systems without, e.g., sequential circuits versus combinational circuits, or machine learning versus adaptive systems. The only difference is that a single macrostep of PTM correspond to the whole computation of TM. PTM in each macrostep reads the contents of its read-only input and work tapes, and produces new contents or the work tape and write-only output tape (somehow in the style of RAM machine). The idea of initiating a macrostep with a non-blank work tape (in contrast to the classical setting where the work tape is also assumed to be blank at the start of computation) plays an essential role in expressiveness of PTMs.

Because PTMs allow unbounded growth of the work tape, their higher expressiveness is based on the *infinity* principle. They do not require an expressive power of oracles in their *interaction*, or *evolution* of dynamically bound inputs.

4.6 Site and Internet Machines

Van Leeuwen and Wiedermann [43] introduced their Site and Internet machines to better capture an inadequate coverage by Turing machines of: interaction of machines (our *interaction*), non-uniformity of programs (our *evolution* of hardware and software), and infinity of operations (our *infinity*).

Site machines are generally interactive Turing machines with advice. Advice forms a limited (but nevertheless more powerful than TM) version of Turing's oracle.

A *Site machine* can be thought as a TM equipped with several input and output ports via which it communicates with its environment by sending and receiving messages. In particular, it may exchange messages with an agent modeled as advice (oracle) noncomputable function. The Site machine starts its computation with empty tapes. It is essentially driven by a standard Turing machine program. At each step the machine reads the symbols appearing at its input ports. At the same time it writes some symbols to its output ports. Based on the current context, i.e. on the symbols read on the input ports and in the "window" on its tapes, and on the current state, the machine prints new symbols under its heads, moves its windows by one cell to the left or to the right or leaves them as they are, enters a new state, and a cycle repeats. A computation of a site machines is unbounded, i.e. it consists of an infinite stream of k-tuples read from k input ports translated

to an infinite stream of l-tuples written to l output ports.

On the other hand an *Internet machine* as its name implies, supposed to be a model of computations performed by computers connected through the Internet. An *Internet machine* consists of a finite but time-varying set of Sites machines. Each machine in the set is identified by its address. The size of the *Internet machine* can grow at most polynomially with time. Within the Internet machine, Site machines work synchronously and communicate by means of message exchanges.

Van Leeuwen and Wiedermann prove that their *Internet machine* is not more expressive than a single *Site machine*. However, both *Site* and *Internet machines* are more powerful than TMs. Generally, a higher expressiveness of Site machines is due to the power of oracles, because infinite operations can be easily modeled by TMs (representing processes that do not terminate, or programs in infinite loops).

Note that van Leeuwen and Wiedermann model evolution (upgrade of hardware and software) by interaction with oracles, and from infinity only infinite streams are allowed. Using our terminology, Site and Internet machines gain higher expressiveness than TMs by employing an *interaction* principle - they get an advice from oracle, and in a very limited sense they refer to *infinity* principle - by allowing infinite strings of messages.

4.7 The π-calculus

There is now a family of algebraic approaches, called process algebras, in which equations describe behavior of composite systems. They include among others Hoare's CSP, Hennessy's Algebraic Theory of Processes (ATP, 1984), Bergstra's and Klop Algebra of Communicating Processes (ACP, 1985), Milner's Calculus of Communicating Systems(CCS, 1980) and π-calculus (Milner, Parrow, Walker, 1989) [4]. Among them the π-calculus [28, 29, 30] is believed to be the most general and basic theory of concurrency for describing interactive processes, i.e., processes that continuously interact with an environment and other processes.

Milner's π-calculus of mobile processes is dynamic extension of the CCS [27]. Like CSP [22] and CCS [27], π-calculus uses a synchronous message passing by handshaking. Both CSP and CCS have static communication channels. The π-calculus allows an arbitrary link topology, and to change it by sending new links through existing ones. The ability to directly represent mobility, in the sense of processes that reconfigure their interconnection structure when they execute, makes it easy to model systems where processes move between different locations and where resources are allocated dynamically.

The π-calculus subsumes the canonical approach to sequential computing; i.e., the Church λ-calculus - the above means that the π-calculus is at least equally expressive as Turing Machines.

The syntax of π-calculus consists of action prefixes π and processes P:

π	::=	$x(y)$	receive name y along channel x
	|	$\bar{x}(y)$	send name y along channel x
	|	τ	unobservable action

P	::=	$\sum_{i\in I}\pi_i.P_i$	summation (choice) of P_i guarded (prefixed) by π_i, in particular, an empty sum (blocking) is denoted by 0
	|	$P_1 \mid P_2$	(parallel) composition
	|	*new a P*	restriction (creation of new private/local a for P)
	|	$!\,P$	replication (as many times as needed) $!\,P = P \mid P \mid ...$

The original (older) version of π-calculus used instead of replication, a defined process call $A(y_1,,y_n)$ and its associated definition $A(x_1,,x_n) = P$ where names $x_1,,x_n$ are distinct and the only names which may occur free in P (not bound by a receive prefix or restriction operators) $A(y_1,,y_n)$ behaves like $P\{y_1/x_1,,y_n/x_n\}$ where $P\{y_i/x_i\}$, $1 \le i \le n$ means the simultaneous substitution of y_i for all free occurrences of x_i.

According to [30] there is yet no definite measure on expressiveness of the π-calculus, although there is strong evidence that its primitives are enough for a wide variety of purposes, including encoding of the functional and object-oriented paradigms.

We justify below that π-calculus might be more expressive than TMs. Everything depends on the interpretation of π-calculus replication (or its predecessor: recursive definition) operator. If the replication is unbounded (infinite) then by the infinity principle, π-calculus is more expressive than TMs. In particular, then it will allow model an infinite number of cells of cellular automata, discrete neural networks, or random automata networks, i.e., models more expressive than TMs. Without infinity, unless to assume that π-calculus processes have the power of oracles, we remain in the class of TMs. Our interpretation is that *as many times as needed* implies the infinite number of replication, in particular. With such interpretation of unbounded replication (or recursive definition, or parallel composition), we obtain by the principle of *infinity* that π-calculus is more expressive than TMs.

4.8 The $ -calculus

The $-calculus is a process algebra of Bounded Rational Agents for Interactive Problem Solving. It has been introduced in the middle of 1990s [10, 11]. The $-calculus (pronounced cost calculus) is a formalization of resource-bounded computation (called also anytime algorithms), proposed by Dean, Horvitz, Zilberstein and Russell in late 1980s and begin of 1990s [23]. Anytime algorithms guarantee to produce better results if more resources (e.g., time, memory) become available.

It appears that interactive agents require formalisms more expressive than TM. The $-calculus can express formalisms having richer behaviors than Turing Machines, including π-calculus, cellular automata, interaction machines, neural nets, and random automata networks [10]. The $-calculus is applicable to robotics, software agents, neural nets, and evolutionary computation. Potentially could be used for design of cost languages, cellular evolvable cost-driven hardware, DNA-based computing and molecular biology, electronic commerce, and quantum computing.

In $-calculus everything is a cost expression: agents, environment, communication, interaction links, inference engines, modified structures, data, code, and meta-code.

The syntax of $-calculus consists of simple (contract) $-expressions α (considered to be executed in one atomic indivisible step) and composite (interruptible) $-expressions C:

α	::=	$(\$_{i\in I}\ P_i)$	cost
	|	$(\rightarrow_{i\in I}\ a\ P_i)$	send P_i through channel a
	|	$(\leftarrow_{i\in I}\ a\ X_i)$	receive X_i from channel a
	|	$('_{i\in I}\ P_i)$	suppress evaluation of P_i, double suppression causes evaluation of P_i exactly once
	|	$(a_{i\in I}\ P_i)$	call of (user) defined simple $-expression a
	|	$(\bar{a}_{i\in I}\ P_i)$	negation of (user) defined simple $-expression a

C	::=	$(\circ_{i\in I} P_i)$	sequential composition
	|	$(\parallel_{i\in I} P_i)$	parallel composition
	|	$(\uplus_{i\in I}\ P_i)$	cost choice
	|	$(\biguplus_{i\in I}\ P_i)$	adversary choice
	|	$(\sqcup_{i\in I}\ (\circ\ \alpha_i\ P_i))$	general choice
	|	$(f_{i\in I}\ P_i)$	defined process call f with parameters P_i, and its associated definition $(:=\ (f_{i\in I}\ X_i)\ P)$ with body P

The indexing set I is a possibly countably infinite. In the case if I is empty, we write empty parallel composition, general and cost choices as $\perp$ (blocking), and empty sequential composition as ε (invisible transparent action, which is used to mask, make invisible parts of $-expressions). Adaptation (evolution/upgrade) is an essential part of $-calculus, and all $-calculus operators are infinite (an indexing set I is unbounded). The $-calculus agents interact through send-receive pairs as the essential primitives of the model.

Because of the infinity of I, it is clear that $-calculus derives its expressiveness from the *infinity* principle. If to assume that simple $-expressions may represent oracles, then $-calculus may represent *interaction* and *evolution* principles too.

However, it is easier and "cleaner" to think about implementation of unbounded (infinite) concepts, than about implementation of oracles. The implementation of scalable computers (e.g., scalable massively parallel computers or unbounded growth of Internet) allows to think about a reasonable approximation of the implementation of *infinity* (and, in particular, PTMs, π-calculus, or $-calculus). At this point, it is completely unclear how to implement oracles (as Turing stated *an oracle cannot be a machine*, i.e., implementable by mechanical means), and as the result, the models based on them (e.g., Site or Internet machines).

4.9 Discrete Neural Networks and Automata Networks

An infinite supply of neurons and finite state machines is used in Max Garzon's models of discrete neural networks and random automata networks [16, 17]. As we pointed out in section 4.2, cellular automata are more expressive than Turing machines, i.e., there exist problems solvable by cellular automata but not by Turing machines.

(Discrete) neural networks can be thought as an extension of cellular automata. Neural Networks (NN) differ from cellular automata that the underlying (possibly infinite) network is no longer homogeneous but an arbitrary digraph; and the transition functions compute weighted sums of inputs from neighboring cells. Neural networks can be discrete (inputs and outputs are discrete) or continuous (operating on real numbers).

It is possible to simulate TM on a neural network with only two real-value neurons [24] (in the spirit that TM can be simulated by two pushdown automata). A discrete neural network with an infinite number of neurons can solve a halting problem of TM [16, 17]. It is unknown the expressive power of a neural network with an infinite number of real-value neurons.

Automata (Random) Networks (AN), similar to neural networks, operate also on an arbitrary (potentially infinite) digraph where in each node an arbitrary finite state machine is located. Automata Networks can simulate an arbitrary discrete neural network.

It has been established in [17] that cellular automata are more expressive than Turing machines, and that an arbitrary cellular automaton (CA) can be simulated by discrete neural networks (NN), and discrete neural networks can be simulated by random automata networks (AN).

Every self-map $T : C \rightarrow C$ realizable on a cellular automaton can be implemented by some neural network, and every discrete neural network by some random automata network, i.e., $CA \subseteq NN \subseteq AN$.

The expressiveness of automata networks, neural networks as well as cellular automata stems from the *infinity principle*. All these models allow either an

infinite number of finite state machines or artificial neurons. Continuous neural networks lead to analog computation.

4.10 Analog Computation and Real-Value Neural Networks

In the 1940-50s analog and digital computers were equally popular. Gradually analog computers have been squeezed out by digital technology, because of better flexibility, greater precision and noise reduction. However, it looks that digital technology is not superior in everything over the analog one. Analog computers because they operate on analog "precise" physical values are potentially more expressive than their digital counterparts.

The main property that distinguishes analog from digital computational models is the use of a continuous state space [32]. The neural network model is a special case of analog computation. Neural networks are proposed as a standard model for analog computing, functioning in the role parallel to that of the Turing machine in the Church-Turing thesis. Siegelmann formulates an equivalent to Church-Turing thesis for analog/neural network computing: *No possible abstract analog device can have more computational capabilities (up to polynomial time) than first-order (i.e., the net neuron function is a weighted sum of its inputs) recurrent neural networks.*

According to [32] the extra power of neural networks (able to compute nonrecursive functions) stems from their nonrational (i.e., real) weights. Note that Garzon's neural networks operate on an infinite number of discrete neurons, Siegelmann's neural networks have a finite number of neurons taking continuous values in their activation functions. Both types of neural networks utilize the *infinity principle*: either infinite number of processing elements (neurons), or infinite (nonenumerable) computing alphabet.

4.11 Evolutionary Turing Machines

By an *Evolutionary Turing Machine* (ETM) [13], we mean a (possibly infinite) series of Turing Machines, where the outcome tape from one generation forms the input tape to the next generation. In this sense ETM resembles a Persistent Turing Machine. The tape keeps both a population of solutions, and the description of an evolutionary algorithm. Both population of solutions and evolutionary algorithms can evolve from generation to generation. The goal (or halting) state of ETM is represented by the optimum of the fitness performance measure. The higher expressiveness of ETM is obtained either evolving infinite populations, or applying an infinite number of generations (the infinity principle), or applying variation (crossover/mutation) operators producing nonrecursive solutions (the evolution principle).

4.12 Accelerating Universal Turing Machines and Inductive Turing Machines

Other superTuring models include Inductive Turing Machines running super- recursive algorithms [5] and *Accelerating Universal Turing Machines* (AUTMs) [7], where programs are executed at an ever-accelerating rate, permitting infinite programs to finish in finite time. Super-recursive algorithms are not required to halt, however they are required to provide the result in the finite time - something in the style of reactive systems. On the other hand, AUTMs require a maximum of two time units to execute any possible program; for example, the first operation takes 1 time unit, the second 0.5 time unit, the third 0.25, and so on.

5 Four Ways to Deal with the TM Intractability: The Interaction, Evolution, Guessing and Parallelism Principles

The basic ways to approach intractable problems are varied and they employ: nondeterministic, randomized, heuristic, approximation, probabilistic, greedy, anytime, evolutionary, interactive and parallel algorithms.

In a similar way as in section 3, where we identified three principles to extend the TM model to deal with undecidability, we identify four ways to deal with intractability, namely by *interaction, evolution, guessing* and *parallelism*. All these methods try to cut search space, either by narrowing search using interaction with environment or other agents. by transforming a problem to a simpler one, by guessing, or by parallel brute force search.

The four principles to deal with the TM intractability are listed below:

1. **Interaction**: by getting feedback from the external world. The world can be represented by an environment or other agents. Feedback can be in the form of direct advice (represented by interactive algorithms, or interactive Turing test for intelligence), or using some performance measure obtained by interaction (measurement, observation) with environment describing "goodness" of a solution and/or resources used to find it. Performance measures are utilized by evolutionary algorithms, anytime algorithms, A* or Minimax search algorithms, dynamic programming, $-calculus.

2. **Evolution**: transforming problem to a simpler (less complex) or incomplete one. This approach is used by structured programming, approximation algorithms, neural networks (performing approximate classification/regression),

anytime algorithms, and $-calculus. In spite, of their name, evolutionary algorithms, in general, do not simplify problem to solve (excluding, perhaps, Genetic Programming with Automatically Defined Functions [2]

3. **Guessing**: selecting randomly a path to find a solution. This approach is used by nondeterministic, probabilistic, randomized, ergodic, evolutionary algorithms, simulated annealing, random restart hill climbing. The ability to guess is indeed a powerful gift! [17]

4. **Parallelism**: exploring all possible solutions simultaneously by trading usually time complexity for space complexity. This approach is represented by quantum computing, DNA-based computing, neural networks, parallel computing on supercomputers or on Internet, and $-calculus.

Some models may use a combination of a few principles. For example, evolutionary computation uses interaction, guessing and parallelism principles, and $-calculus all four methods.

In the next section, we will present how the above four principles are used by several computational models.

6 Beyond TM Intractability

This section presents some models whose goal is to deal with the intractability of hard computational algorithms rather than to attempt to solve nonrecursive problems.

6.1 Quantum Computing

Quantum Computing [15, 9] is the natural extension of classical computing to the processing at the atomic scale using individual atoms, molecules or photons. Breaking supposedly "unbreakable" codes, teleportation, generation of true random numbers is based on quantum superposition principle that photon, atom or electron can be simultaneously in many states. However, the attempt to "read" those states always finds a quantum element in one specific state.

The problem with a classical TM model using a single tape is that it allows to capture only sequential computation. We need either multitape (or multitrack) TMs or to extend an alphabet of TM to represent a set of symbols. This second approach was used by David Deutsch in his *Quantum Turing Machine QTM* [9]. In his binary QTM, he extended a binary alphabet {0, 1} of the Turing Machine[1]

[1] of course, we can do a similar extension for any n-element alphabet

by adding a third symbol, representing a blend, or "superposition", of 0 and 1 simultaneously. Thus the QTM had the potential for encoding many inputs to a problem simultaneously on the same tape, and performing a calculation on all the inputs in the time it took to do just one of the calculations classically. This technique was dubbed "quantum parallelism". We can think about quantum bits (called "qubits") as vectors with coordinates x and y (representing 0 and 1) and quantum computing as processing vectors (i.e., both coordinates simultaneously). However, our measuring equipment is calibrated (is able) to read only either x or y coordinate. Destructive process of reading, projects a vector either to x or y coordinate, because our measurement apparatus allows only to read values with some discrete precision (similar like analog values can be read only with some precision), thus as the result 0 or 1 is read.

As was observed by Siegelmann [32], both quantum computing and DNA-based computing are conjectured to compute in polynomial time recursive functions which are not in P (polynomial). However, they cannot compute nonrecursive functions, i.e. they are not more expressive than Universal TM. It is clear that quantum computing applies the *parallelism* principle to cut the search time for a solution, but is not more expressive than TM.

However, if to apply quantum parallelism to superTuring models of computation, we will obtain quantum superTuring models more expressive than TM.

For example, we can define a superTuring *Quantum Cellular Automaton* with neighborhood of n cells. Then each "quantum cell" can be modeled by 2^{n+1} "ordinary" cells, representing a different and unique assignment of 0 and 1 to a cell and its neighbors. Each of 2^{n+1} cells compute outputs simultaneously. However, our reading allows us to read state of only one cell at the time, and after destructive reading we cannot read states of remaining cells. We may have a probabilistic choice, which cell to pick up for reading. The true difference between *Quantum Cellular Automaton* QCA and QTM is that QCA is more expressive that QTM or TM - it has an infinite number of quantum cells, and by the *infinity* principle it is more expressive than TM. Thus Quantum Cellular Automaton, dissimilar to Quantum Turing Machine, belongs to superTuring models of computation. The notion of quantum computing is orthogonal to the expressiveness of computational models. If it is applied to the model more expressive than Turing Machine, we obtain a quantum superTuring model, otherwise we stay in the class of Turing Machines.

6.2 DNA-Based Computing

Biomolecular computing, and in particular, DNA-based computing is another promising technology. The notion of using DNA molecules to do computing originated in a paper by Adleman [1] who solved the NP-complete Hamiltonian path

problem in a test tube. Lipton showed how to use the same primitive DNA operations as Adleman to solve directly any SAT problem [25] and Beaver proved that the class of problems solved in a feasible way using Adleman's methods is PSPACE [3]. Both Beaver and independently Smith [3, 33] showed how to construct a molecular nondeterministic Turing machine, i.e. that molecular computers are Turing computation universal.

Scientists showed how using Adleman's method to solve NP-complete problems, how to do arithmetic, or simulate finite state machines. DNA-based computing because of its miniaturization and natural superparallelism, it can be useful at least in the solution of search problems. However, all the experiments were on the level of the test tube so far, and nobody built a biomolecular computer, or even biochip yet (however, DNA chip arrays are manufactured).

DNA-based computing in the Adleman's style implements the class of massively parallel nondeterministic algorithms. It consists of four basic operations: *encoding* that maps the problem onto alphabet of nucleic acid bases $\{A, T, G, C\}$, *hybridization* and *ligation* which perform the basic core processing, and *extraction* that makes the results available for humans. *Hybridization* is a chemical process that joins two complementary single strands into a double strand such that A binds with T and C with G. *Ligation* is a chemical process where two double strands are joined into one double strand. *Extraction* may use PCR techniques.

DNA-based computing uses both *parallelism* and *guessing*, i.e., many millions of DNA chains compute simultaneously checking an enormous number of randomly selected possibilities. This is parallel and nondeterministic (but not necessarily exhaustive, i.e. checking all possible solutions) search.

6.3 Evolutionary Computation

Evolutionary Computation (EC) [2] is a relatively new approach to computer problem solving EC is based on principles of natural selection (genetic methods, fitness driven). Evolution is a two-step process of random variation and natural selection: variation creates diversity, which is heritable, and selection eliminates inappropriate individuals. Evolutionary computation has been invented as many as 10 times independently from 1953-1970. Alan Turing proposed already to use a kind of genetic algorithm in his unorganized machines [36].

EC consists of four main subareas *Genetic Algorithms, Genetic Programming, Evolution Strategies, and Evolutionary Programming*. It can be understood as a probabilistic beam search directed by fitness optimization to solve hard optimization problems.

Evolutionary Algorithm can be described in the form of the functional equation working in a simple iterative loop in discrete time t:

$$\vec{x}[t+1] = s(v(\vec{x}[t])), \textit{where}$$

$\vec{x}$ - is a population under a representation, s - is selection operator (e.g., truncation, proportional, tournament), v - is variation operator (e.g., variants of mutation and crossover), $\vec{x}[0]$ - is initialization of population.

In the above equation only the termination condition is omitted. The true termination condition is to find a population with a global maximum fitness of one of individuals in the population. Usually, this is not possible to achieve, and EC terminates after a predefined number of iterations, or when no more improvements are observed through several generations.

Evolutionary computation in a general case is incomplete (simply probabilistic search can miss some solutions) and not optimal (if some solutions are missing, then they might be exactly the optimal ones). Under some conditions is can be complete and optimal. However, evolutionary computation, even if optimal, is not totally optimal, i.e., it is computationally very expensive, with missing mechanisms for control search cost (which would allow scalability and evolution on-line, e.g., for evolutionary robotics).

In the solution of hard computation problems, evolutionary computation uses *interaction* principle (fitness measures a solution quality in a given environment), *guessing* principle (search points are selected randomly with bias towards fitter individuals), and *parallelism* principle (a whole population of solutions (co-existing in parallel in the same environment) is tested in one generation).

6.4 Anytime Algorithms

Anytime Algorithms [23] attempt to find the best answer possible given operational constraints. They are known also as resource-bounded computation, just-in time computing, flexible computation, imprecise computation, design-to-time scheduling, decision-theoretic metareasoning. They guarantee better results if more time or memory is available for deliberation. Anytime Algorithms trade off the quality of solutions for the amount of resources used.

A *bounded rational agent* is one that always takes the action that is expected to optimize its performance measure, given the percept sequence it has seen so far and limited resources one has. A bounded rational agent solves constraint optimization problems, taking into account both quality of a solution and costs of resources used to obtain it. Bounded optimality fits intuitive idea of intelligence and provides a bridge between theory and practice.

Intelligence of an agent can be judged on the basis of cleverness of its search strategy/algorithm for the best (least costly) solution. Search algorithms are evaluated on the basis of their

- *Completeness:* is the strategy guaranteed to find a solution when there is one?
- *Search Cost:* how much resources (e.g., how long does it take and how much memory) are used to find a solution?
- *Optimality:* does the strategy find the highest quality solution when there are several different solutions?
- *Total Optimality:* does the strategy find the highest quality solution with optimal search cost?

Search can be *cooperative*, *competitive*, or *random* and involve single, pairs or multiple agents. In cooperative search other agents help to find an optimum, in competitive search - they distract to reach an optimum, and in random search other agents do not care about helping or distracting to reach an optimum.

Anytime Algorithms apply the *interaction* and *evolution principles* to deal with intractability. They use the utility performance profiles to measure candidate solutions performance, and to control search cost - the *interaction* principle. The solutions in interruptible anytime algorithms are incomplete and they are improved from iteration to iteration - the *evolution* principle.

6.5 The $ -calculus

We presented the $-calculus process algebra of Bounded Rational Agents in section 4.8. In this section, we will show how $-calculus is equipped to deal with intractability.

The basic $-calculus search method used for problem solving is called the $k\Omega$-optimization. It is a very general search method, allowing to simulate many other search algorithms, including A*, Minimax, dynamic programming, tabu search, or evolutionary algorithms. The problem solving works iteratively: through select, examine and execute phases. In the *select phase* the tree of possible solutions is generated up to k steps ahead, and agent identifies its alphabet of interest for optimization Ω. This means that the tree of solutions may be incomplete in width and depth (to deal with complexity). However, incomplete (missing) parts of the tree are modeled by silent $-expressions ε, and their cost estimated (i.e., not all information is lost). The above means that the $k\Omega$-optimization can be complete and optimal if some conditions are satisfied. In the *examine phase* the trees of possible solutions are pruned minimizing cost of solutions, and in the *execute phase* up to n instructions are executed. Moreover, because the $ operator may capture not only the cost of solutions, but the cost of resources used to find a

solution, we obtain a powerful tool to avoid methods that are too costly, i.e., $-calculus directly minimizes search cost. This basic feature, inherited from anytime algorithms, is needed to tackle directly hard optimization problems, and allows to solve total optimization problems (the best quality solutions with minimal search costs).

In fact, to deal with intractability, $-calculus employs all four principles: *interaction* - it interacts with other agents and environment to obtain costs representing the quality of solutions and search cost; *evolution* - as in anytime algorithms solutions may be incomplete and are built incrementally. Additionally, because of recursive definition operator, $-calculus may use procedural abstraction (the old idea from structured programming) to cut a search space; *quessing* - using general choice operator, instead of cost choice operator allows to perform random guesses in search for a solution; *parallelism* - $-calculus is inherently parallel model. This means that multiple search points can be explored simultaneously.

7 Conclusions

It is interesting that models dealing with the TM undecidability are either very old (i.e., from 1930-1950s, e.g., o-machines, c-machines, u-machines, cellular automata, neural networks, analog computation) or quite new (i.e., from 1990-2000s, e.g., Interaction Machines, Persistent Turing Machines, π-calculus, $- calculus, Site and Internet Machines). There is nothing between. This means then the algorithmic approach[2] was very efficient to suppress all research on non-algorithmic models of computation in 1960-1980s.

On the other hand, models dealing with intractability are quite new, i.e., from 1980-1990s. This applies to all models presented in section 6, including evolutionary computation, which was re-discovered more than 10 times in 1950-1970s, but started to bloom really in 1990s.

Two principles, *interaction* and *evolution* are shared both by models dealing with undecidability and with intractability. However, there are essential differences how these principles are used. For example, opening the closed model by *interaction* is used to increase expressiveness (by getting an advice from the external component) in models dealing with undecidability, or to get the performance measure from the external component (an environment or other agents) to decrease the complexity of search for solutions for models attacking intractability. Similarly, the *evolution* principle in the first instance, is used to adapt solution to the more expressive one, and in the later case, to simplify it. The *infinity* principle increases expressiveness, but does not help to decrease search costs. On the

[2] by the way, very elegant and useful - unfortuntaly, not covering all possible forms of computation

other hand, neither *parallelism* (unless unbounded) nor *guessing* are sufficient for higher expressiveness. However, parallelism is a necessary prerequisite for interaction (there have to coexist in parallel at least two components to interact). The interplay between the above principles to control expressiveness and efficiency has not been investigated yet to any extent.

To be effective, models controlling intractability, use several principles simultaneously. However, only Interaction Machines apply all three principles to deal with unsolvable problems. It is unclear, how combination of several principles, may increase expressiveness of such models. We know that the application of one principle only (e.g., the *interaction*, *evolution* or *infinity*) is sufficient to derive models more expressive than TMs. We do not know, whether a combination of two or three such principles leads to a hierarchy of models going beyond TMs. Some preliminary steps have been done in that direction, i.e., the establishment of hierarchy of expressiveness for cellular automata, discrete, neural networks, automata networks, and the relation between Sequential and Multistream Interaction Machines [17, 38, 10], but much more research is needed.

References

[1] Adleman L.M., Molecular Computation of Solutions to Combinatorial Problems, Science, vol.266, 1994, 1021-1024.

[2] Bäck T., Fogel D.B., Michalewicz Z. (eds.), Handbook of Evolutionary Computation, Oxford University Press, 1997.

[3] Beaver D., A Universal Molecular Computer, DNA Based Computers, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol.27, 1996, 29-36.

[4] Bergstra J.A., Ponse A., Smolka S.A. (eds.), Handbook of Process Algebra, North Holland, Elsevier, 2001.

[5] Burgin M., How We Know What Technology Can Do, *Communications of the ACM* 44:11, Nov. 2001, 83-88.

[6] Burks A., Essays on Cellular Automata, Univ. of Illinois Press, 1970.

[7] Copeland B.J., Super-Turing Machines, *Complexity*, 4(1), 1998, 30-32.

[8] Church A., An Unsolvable Problem of Elementary Number Theory, American Journal of Mathematics, vol.58, 1936, 345-363.

[9] Deutsch D., Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer, Proc. of the Royal Society of London, vol.400, 1985, 97-117.

[10] Eberbach E., Expressiveness of $-Calculus: What Matters?, in: Advances in Soft Computing, Proc. of the 9th Intern. Symp. on Intelligent Information Systems IIS'2000, Bystra, Poland, Physica-Verlag, 2000, 145-157.

[11] Eberbach E., $-Calculus Bounded Rationality = Process Algebra + Anytime Algorithms, in: (ed.J.C.Misra) Applicable Mathematics: Its Perspectives and Challenges, Narosa Publishing House, New Delhi, Mumbai, Calcutta, 2001, 213-220.

[12] Eberbach E., Is Entscheidungsproblem Solvable? Beyond Undecidability of Turing Machines and Its Consequence for Computer Science and Mathematics, in: (ed.J.C.Misra) Computational Mathematics, Modeling and Algorithms, Narosa Publishing House, New Delhi, 2003, 1-32.

[13] Eberbach E., On Expressiveness of Evolutionary Computation: Is EC Algorithmic?, *Proc. 2002 World Congress on Computational Intelligence* (WCCI), Honolulu, HI, 2002, pp. 564-569.

[14] Eberbach, E., Goldin, D., Wegner, P., Turing's Ideas and Models of Computation, in: (ed.Teuscher, Ch.), Alan Turing: Life and Legacy of a Great Thinker, Springer-Verlag, Berlin, Heildelberg, New York, 2003.

[15] Feynman R., Simulating Physics with Computers, Intern. Journal of Theoretical Physics, vol.21, no.6/7, 1982, 467-488.

[16] Garzon M., Franklin S., Neural Computability, Proc. Third Intern. Joint Conf. on Neural Networks, vol.2, 1989, 631-637.

[17] Garzon M., Models of Massive Parallelism: Analysis of Cellular Automata and Neural Networks, An EATCS series, Springer-Verlag, 1995.

[18] Gödel K., *Über* formal unedtscheidbare *Sätze* der Principia Mathematica und verwandter Systeme, I, Monatshefte *für* Mathematik und Physik, vol.38, 1931, 173-198.

[19] Gödel K., Collected Works Volume III: Unpublished Essays and Lectures, S.Feferman et al., eds., Oxford Univ. Press, 1995.

[20] Goldin D., Persistent Turing Machines as a Model of Interactive Computation, FoIKS'00, Cottbus, Germany, 2000.

[21] Goldin D., Smolka S., Wegner P., Turing Machines, Transition Systems, and Interaction, 8th Int'l Workshop on Expressiveness in Concurrency, Aarlborg, Denmark, August 2001.

[22] Hoare C.A.R., Communicating Sequential Processes, Prentice-Hall, 1985.

[23] Horvitz E., Zilberstein S. (eds.), Computational Tradeoffs under Bounded Resources, Artificial Intelligence, 126, 2001, 1-196.

[24] Koiran P., Cosnard M., Garzon M., Computability with Low-Dimensional Dynamical Systems, Theoretical Computer Science, 132 (1-2), Sept. 1994, 113-128.

[25] Lipton R., DNA Solutions of Hard Computational Problems, Science, vol.268, 1995, 542-545.

[26] McCulloch W., Pitts W, A Logical Calculus of the Ideas Immanent in Nervous Activity, Bulletin of Mathematical Biophysics, vol.5, 1943, 115-133.

[27] Milner R., A Calculus of Communicating Systems, Lect. Notes in Computer Science vol.94, Springer-Verlag, 1980.

[28] Milner R., Parrow J., Walker D., A Calculus of Mobile Processes, I & II, Information and Computation 100, 1992, 1-77.

[29] Milner R., Elements of Interaction, CACM, vol.36, no.1, Jan. 1993, 78-89.

[30] Plotkin G., Stirling C., Tofte M. (eds.), Proof, Language, and Interaction: Essays in Honour of Robin Milner, The MIT Press, 2000.

[31] Rosenblatt F., The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Psychological Review, vol.65, 1958, 386-408.

[32] Siegelmann H., Neural Networks and Analog Computation: Beyond the Turing Limit, Birkhauser, 1999.

[33] Smith W.D., DNA Computers in Vitro and Vivo, DNA Based Computers, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol.27, 1996, 121-185.

[34] Turing A., On Computable Numbers, with an Application to the Entscheidungsproblem, Proc. London Math. Soc., 42-2, 1936, 230-265; A correction, ibid, 43, 1937, 544-546.

[35] Turing A., Systems of Logic based on Ordinals, Proc. London Math. Soc. Series 2, 45, 1939, 161-228.

[36] Turing A., Intelligent Machinery, in Colected Works of A.M. Turing: Mechanical Intelligence, ed.D.C.Ince, Elsevier Science, 1992.

[37] Wegner P., Why Interaction is More Powerful Than Algorithms, CACM, May 1997, vol.40, no.5, 81-91.

[38] Wegner P., Interactive Foundations of Computing, Theoretical Computer Science, 192, 1998, 315-351.

[39] Wegner P., Goldin D. (1999) Coinductive Models of Finite Computing Agents, Electronic Notes in Theoretical Computer Science, vol.19.

[40] Wegner, P., Goldin, D., Computation Beyond Turing Machines: Seeking Appropriate Methods to Model Computing and Human Thought, CACM, vol.46, no.4, 2003, 100-102.

[41] Wegner P., Eberbach E., New Models of Computation, The Computer Journal, 47(1), The British Computer Society, Oxford University Press, 2004.

[42] Whitehead A.N., Russell B., Principia Mathematica, vol.1, 1910, vol.2, 1912, vol.3, 1913, Cambridge Univ. Press, London.

[43] Van Leeuwen J., Wiedermann J., The Turing Machine Paradigm in Contemporary Computing, in. B. Enquist and W. Schmidt (eds.) Mathematics Unlimited - 2001 and Beyond, LNCS, Springer-Verlag, 2000.

[44] Von Neumann J., Theory of Self-Reproducing Automata, (edited and completed by Burks A.W.), Univ. of Illinois Press, 1966.

Building Reliable Systems from Unreliable Components and the DNA Computing Paradigm

Elena Losseva †

Abstract

An old problem of building reliable systems from unreliable components is revisited in the context of DNA computing. Taking finite automata as a model, the effect of basic language operations on reliability of automata is investigated.

1 Motivation

It became apparent in the early stages of computing that learning how to build reliable systems from unreliable components is a task of fundamental importance. Already in 1956 J. von Neumann writes that an error should be viewed "not as an extraneous and misdirected or misdirecting accident, but as an essential part of the process under consideration" [9]. These words are more than relevant for DNA computing. While it is difficult to predict the exact nature of DNA computing as it might come into practical existence, one thing is clear - errors are unavoidable.

Now imagine yourself being transported into some distant future, where biocomputing dreams start to become a reality. Perhaps it is possible for you to take a description of a small Turing machine to your biochemist friend's wetlab and have it implemented in DNA the same morning. However one day, when you are feeling particularly ambitious and bring for implementation a TM of somewhat grandiose proportions, your friend shakes his head and says that he cannot build a TM of that size. Being in an adventurous mood however, he is willing to try to build the small components of the TM, and after some combination of concatenation, union, and Kleene star to obtain the desired machine. The problem unfortunately, is that the already questionable reliability of the implemented components is likely to be unpredictable for the overall machine.

† Department of Computer Science, University of Western Ontario, London, Ontario, Canada N6A 5B7 (elena@csd.uwo.ca)