

Expressiveness of the π -Calculus and the $\$$ -Calculus

Eugene Eberbach

Computer and Information Science Dept.

University of Massachusetts

North Dartmouth, MA 02747-2300

Email: eeberbach@umassd.edu

Abstract: *In this paper we investigate the expressiveness of two process algebras, the π -calculus of mobile processes and the $\$$ -calculus of bounded rational agents. We demonstrate that both models are more expressive than Turing Machines, i.e., they belong to super-Turing models of computation. In particular, they are able to solve the halting problem of the Universal Turing Machine. Additionally, the $\$$ -calculus can approximate the solution of the universal search algorithm, and can simulate the π -calculus, i.e., it is at least equally expressive as the π -calculus.*

Keywords: *process algebras, expressiveness, superTuring models of computation, π -calculus, $\$$ -calculus*

I. Introduction

Turing Machines [23], [24] provide the fundamental model for sequential computing. On the other hand, process algebras [1] are considered as the most mature approach to concurrent computing. In this paper, expressiveness of two process algebras are investigated. These are the π -calculus process algebra of mobile processes [16], [17], [18], [20], [19], [22], and the $\$$ -calculus process algebra of bounded rational agents [4], [6], [8], [9], [10], [25].

Turing Machines and algorithms are two fundamental concepts of computer science and problem solving. Turing Machines describe the limits of algorithmic problem solving, and laid the foundation of current computer science in the 1960s. It turns out that *undecidable problems* cannot be solved by TMs and *intractable problems* are solvable, but require too many steps. For undecidable problems effective recipes do not exist - problems are called nonalgorithmic or nonrecursive. On the other hand, for intractable problems algorithms exist, but running them on a deterministic Turing Machine, requires an exponential amount of time (the number of elementary moves of the TM) as a function of the TM input.

An algorithm should consist of a finite number of steps, each having well defined and implementable meaning. We are convinced that computer computations are not restricted to algorithms only.

Definition 1: *By superTuring computation we mean any computation that cannot be carried out by a Turing Machine as well as any (algorithmic) computation carried out by a Turing Machine.*

In [8], [9], [25], several superTuring models have been discussed and overviewed. The incomplete list includes Turing's o-machines, c-machines and u-machines, cellular automata, discrete and analog neural networks, Interaction Machines, Persistent Turing Machines, Site and Internet Machines, the

π -calculus, the $\$$ -calculus, Inductive Turing Machines, Infinite Time Turing Machines, Accelerating Turing Machines and Evolutionary Turing Machines.

SuperTuring models derive their higher than the TM expressiveness using three principles: *interaction*, *evolution*, or *infinity*. In the *interaction principle* the model becomes open and the agent interacts with either a more expressive component or with an infinite many components. In the *evolution principle*, the model can evolve to a more expressive one using non-recursive variation operators. In the *infinity principle*, models can use unbounded resources: time, memory, the number of computational elements, an unbounded initial configuration, an infinite alphabet, etc. The details can be found in [8], [9], [25].

The paper is organized as follows. In section 2, we briefly outline the π -calculus process algebra of mobile processes. In section 3, the same is done for the $\$$ -calculus process algebra of bounded rational agents. In section 4 and 5, the expressiveness of the π -calculus and the $\$$ -calculus are investigated, respectively. Section 6 contains conclusions.

II. The π -Calculus Process Algebra of Mobile Processes

The Milner's π -calculus [16], [17], [18], [20], [19], [22] is a mathematical model of processes whose interconnections change as they interact. It is a model of the changing connectivity of *interactive systems*. In such a sense the π -calculus of mobile processes can be considered to be a dynamic extension of CCS (Calculus of Communicating Systems) [15]. Like CSP [11] and CCS [15], the 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 can be seen as a basic model of computation [18]. Every basic model rests upon a small number of primitive notions; the π -calculus rests upon the primitive notion of *interaction*, just as Turing Machines rest upon the notion of reading and writing a storage medium, and just as recursive equations and the λ -calculus rest upon mathematical functions. The π -calculus subsumes the canonical approach to sequential computing; i.e., the Church λ -calculus [2], [3] - the above

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

The indexing set I is possibly countably infinite. In the case when 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). $\$$ -calculus agents interact through send-receive pair as the essential primitives of the model.

Sequential composition is used when $\$$ -expressions are evaluated in a textual order. Parallel composition is used when expressions run in parallel and it picks a subset of non-blocked elements at random. Cost choice is used to select the cheapest alternative according to a cost metric. Adversary choice is used to select the most expensive alternative according to a cost metric. General choice picks one non-blocked element at random. General choice is different from cost and adversary choices. It uses guards satisfiability. Cost and adversary choices are based on cost functions. Call and definition encapsulate expressions in a more complex form (like procedure or function definitions in programming languages). In particular, they specify recursive or iterative repetition of $\$$ -expressions.

Simple cost expressions execute in one atomic step. Cost functions are used for optimization and adaptation. The user can use built-in standard cost function, or is free to define his/her own cost metrics. Send and receive perform handshaking message-passing communication, and inferencing. Additionally, a user is free to define her/his own simple $\$$ -expressions, which may or may not be negated.

In this paper we will omit the operational semantics of the $\$$ -calculus based on the $k\Omega$ -optimization meta-search. We will skip also definitions of cost performance measures.

IV. The π -Calculus Expressiveness: its Support for Undecidability

According to [20] 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. Robin Milner has proved that the π -calculus is at least equally expressive as Turing Machines, and, additionally, suggested that it is “more complete” than Turing Machines. For the last assumption to be correct, it is not enough to simulate λ -calculus or Turing Machines in π -calculus. The π -calculus has to be more expressive than TMs.

We justify below that the π -calculus might be more expressive than TMs. For example, it is enough to allow the infinity in the π -calculus replication (or its predecessor: recursive definition) operator, i.e., to permit for an infinite replication. Note that replication $!P$ is an abbreviation for an infinite number of copies of P in parallel [18], [22]. Replication is axiomatized by the structural congruence axiom $!P \equiv P \mid !P$. Then if it is solved as the greatest fixed point $!P = P^\infty \cup \bigcup_{i=1}^\infty P^i$, where P^i is a shorthand for P replicated in parallel i -times, it makes possible to express infinite behaviors.

With infinite replication (or infinite recursive call) by the infinity principle, the π -calculus becomes more expressive than TMs. In particular, then it allows model an infinite number of cells of cellular automata, discrete neural networks, or random automata networks in the style of [5], i.e., to simulate models more expressive than TMs. Without infinity, the π -calculus could not simulate cellular automata with the infinite number of cells. With the replication operator allowing for an infinite replication, the π -calculus is able to simulate cellular automata or discrete neural networks, and not merely TMs. Without infinity, unless to assume that the π -calculus processes have the power of oracles, we remain in the class of TMs. With such unbounded replication (or recursive definition, or parallel composition), we obtain by the principle of *infinity* (see introduction and [8], [9], [25]) that the π -calculus is more expressive than TMs. Note that the π -calculus allows in infinity to solve the halting problem of the TM, similar as the $\$$ -calculus does (see section 5), and not only to simulate TMs.

V. The $\$$ -Calculus Expressiveness: its Support for Undecidability

To deal with undecidability, the $\$$ -calculus may represent all three principles from introductory section: the infinity, interaction, and evolution principles:

- *infinity* - because of the infinity of the indexing set I in the $\$$ -calculus operators, it is clear that the $\$$ -calculus derives its expressiveness mostly from the *infinity* principle.
- *interaction* - if to assume that simple $\$$ -expressions may represent oracles, then the $\$$ -calculus can represent the interaction principle. Then we define an equivalent of the oracle as a user defined simple $\$$ -expression, that somehow in the manner of the “black-box” solves unsolvable problems (however, we do not know how).
- *evolution* - the $k\Omega$ -optimization may be evolved to a new (and hopefully) more powerful problem solving method

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, the π -calculus, or the $\$$ -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 [26]).

A. Expressing Sequential and Concurrent Algorithms

The expressiveness of the $\$$ -calculus is not worse than the expressiveness of Turing Machines.

Theorem 1: (On expressing an arbitrary algorithm in the $\$$ -calculus) The $\$$ -calculus can express an arbitrary algorithm. Proof: To prove that, it is enough to show how to encode in the $\$$ -calculus any of the models equivalent to Turing Machines. For example, the λ -calculus is equivalent to Turing Machines and all algorithms. Encoding of the λ -calculus by the subset of the $\$$ -calculus is straightforward:

<u>λ-calculus</u>	<u>$\\$-calculus encoding</u>
x - variable	x - variable name
MN - application	$(f N)$ - function call
$\lambda x.M$ - abstraction	$(:= (f x) M)$ - function definition

Operational semantics:

$(\lambda x.M) N \rightarrow M[N/x]$	$(f N) \rightarrow M[N/x]$,
function application	where $(:= (f x) M)$
$(\beta$ -reduction)	

In [5] it has been demonstrated, how some other models of computation, more expressive than Turing Machines, can be simulated in the $\$$ -calculus. This includes the π -calculus, Interaction Machines, cellular automata, neural networks, and random automata networks. The $\$$ -calculus can encode the π -calculus and this is stated by the theorem below.

Theorem 2: (On simulating the π -calculus in the $\$$ -calculus) The $\$$ -calculus is at least equally expressive as the π -calculus.

Proof: The proof is by encoding of the π -calculus in the $\$$ -calculus

<u>π-calculus (πC)</u>	<u>$\\$-calculus ($\\$C$) encoding</u>
0 - inert process	\perp - zero (blocking)
$x(y).P$ - input prefix	$(\sqcup (\circ (\leftarrow x y) P))$ - receive $(\circ (in\ x\ y) P))$
$\bar{x}y.P$ - output prefix	$(\sqcup (\circ (\rightarrow x y) P))$ - send $(\circ (out\ x\ y) P))$
$P + Q$ - sum	$(\sqcup P Q)$ - general choice
$P \mid Q$ - parallel composition	$(\parallel P Q)$ - parallel composition
$(\nu x) P$ - restriction	$(block_in_out\ x\ P)$
$!P$ - replication	$(:= (!P) (\parallel P (!P)))$

Simulation requires introduction of (user defined) operators: in , out and $block_in_out$, but the $\$$ -calculus allows to do it. Then the input prefix is simulated by the general choice either to do $\$$ -calculus receive (requiring synchronization with corresponding send) or by a new operator $(in\ x\ y)$ working like receive, but not requiring synchronization neither with send nor out. In a similar way, the output prefix is simulated by general choice of send and a new operator out . The π -calculus restriction operator is simulated by a new operator $block_in_out$ blocking execution of in and out operators for a given channel x (and this is a general idea of the π -calculus

restriction operator). Simulation of the replication operator by recursive definition is straightforward.

If to not be too picky about the necessity to use new (user) defined operators in , out , and $block_in_out$, then the π -calculus could be claimed to be a subcalculus of the $\$$ -calculus, because each operator of the π -calculus is simulated by a corresponding operator from the $\$$ -calculus. We have the choice to encode numerals, true, false boolean values, if-then-else operator either as in the λ -calculus or as in the π -calculus, but we prefer the third way: \perp is used to denote *false* (similar to implementation of the negation by failure in Prolog, where exactly one of P or $\neg P$ will be blocked (will fail, i.e., return \perp), and if-then-else we can express easily by general choice with positive and negated guards.

B. Some Undecidability Results

We will present two theorems that the optimization problem for the $\$$ -calculus cannot be solved neither by the Universal Turing Machine nor by the $\$$ -calculus itself.

Every optimization problem for the $\$$ -calculus can be expressed as a halting problem: to stop when the optimum is reached, or to continue computation otherwise.

Theorem 3: (On undecidability of the optimization problem for the $\$$ -calculus by the Universal Turing Machine) The optimization (halting) problem of the $\$$ -calculus is algorithmically unsolvable by the Universal Turing Machine.

Proof: A Universal Turing Machine cannot solve its own halting problem, and the $\$$ -calculus by Theorem 1 can express an arbitrary algorithm/TM being a subset of all possible $\$$ -expressions. If UTM cannot decide about halting (reaching the goal state) for the subset of $\$$ -expressions simulating the λ -calculus, then the same applies to the complete $\$$ -calculus. Another argument confirming the undecidability result is that the reaching the optimum may require an unbounded (infinite) number of steps by the $k\Omega$ -search, thus that problem is algorithmically unsolvable.

We will show that the optimization problem cannot be decided by the $\$$ -calculus itself if we consider all possible search algorithms and all possible problems to solve.

The proof of undecidability will be by the diagonalization technique leading to a contradiction, i.e., by attempting to construct a program finding the universal search algorithm looking for the optimum, and to fail in that. The idea is the following: if a best search algorithm existed, how would it perform having itself as an input?

Theorem 4: (On undecidability of the optimization problem for the $\$$ -calculus by the $\$$ -calculus) The problem of finding the best search algorithm finding the optimum of the arbitrary problem is algorithmically undecidable in the $\$$ -calculus.

Proof: By contradiction. Suppose that there were such a meta-system $k\Omega[t] = A$ that detects for an arbitrary input $x[t]$ whether or not its optimum $\$_3(x[t])$ has been found, and it replies yes or no. Assumption: A takes as an input x and its input file i (can be empty), and decides yes or no that cost function $\$_3(x[t])$ reached its optimum.

Step 1: Modify $k\Omega[t] = A$, called $A1$, that acts like A , but

when A prints no, A_1 prints “optimum reached”.

Step 2: Modify A_1 to A_2 . This program takes only one input, x , and acts like A_1 with both its program and data inputs equal to x , i.e., $A_2(x) = A_1(x, x)$ (A_2 simulates A_1 , storing copy of x to be used instead of i , when A_1 performs input) Note, we repeat essentially what Alan Turing did to prove undecidability of TMs: to modify A to take as its inputs program x , with its input being x itself, and ask what would modified A reply: yes or no? There are somehow related problems: self-modifying programs, bootstrapping, compiling compiler, self-reproduction, carrying yourself, etc.

Step 3: Put A_2 as input to A_2 . A_2 cannot exist. If it did what would $A_2(A_2)$ do?

If $A_2(A_2) = \text{yes}$, then A_2 given A_2 as input evidently does not print “optimum reached” (it prints “yes”). But $A_2(A_2) = A_1(A_2, A_2) = A(A_2, A_2)$, and A_1 prints yes if and only if its first input (program A_2), given its second input as data (program A_2), prints “optimum reached”. Thus $A_2(A_2) = \text{yes}$, implies A_2 prints optimum reached. But if A_2 prints optimum reached, then $A_1(A_2, A_2) = \text{optimum reached}$, and $A(A_2, A_2) = \text{no}$. Thus $A_2(A_2) = \text{optimum reached}$, implies $A_2(A_2)$ does not print “optimum reached”. $A(A_2, A_2)$ cannot at the same time decide yes and no, and A_2 cannot print and not print “optimum reached” at the same time. Thus A_2 does not exist, and as a consequence, A_1 , and A . A cannot decide one way or another - the problem is “undecidable”, i.e., the program (if it existed) for some inputs (which is the deciding program itself), would not be able to reply clearly: yes or no. This has no real sense: the program should answer yes or no, pending that it exists. Thus the only available option left is that such program does not exist.

The proof of the algorithmic undecidability of finding the best search algorithm can be done alternatively by a direct reduction from the language of the Universal Turing Machine or by reduction from the Post Correspondence Problem [21].

It is interesting that the $\$$ -calculus can solve in the limit the halting problem of the Universal Turing Machine, and approximate the solution of the halting/optimization problem of the $\$$ -calculus. This is a very interesting result, because, if correct, besides the $\$$ -calculus, it may suggest that a self-modifying program using infinitary means may approximate the solution of its own decision (halting or optimization) problem.

C. Solving Nonalgorithmically the Turing Machine Halting Problem

The theorems from this section justify that the $\$$ -calculus is more expressive than the TM, and may represent non-algorithmic computation. We will show three ways how the $\$$ -calculus can solve nonalgorithmically the halting problem of the Universal Turing Machine using either *infinity*, *evolution*, or *interaction principles* (defined in introduction).

Theorem 5: (On solution of the halting problem of UTM by the $\$$ -calculus using infinity principle) The halting problem for the Universal Turing Machine is solvable by the $\$$ -calculus using the infinity principle.

Proof: The proof uses the solution of the halting problem of the UTM as the solution in the limit by the $k\Omega$ -search. We will assume the infinite enumerable number of agents A_1, A_2, \dots . Each i -th agent will take as its input a specific instance of the Turing Machine, a pair $(M_i[0], x_i[0])$ where $M_i[0]$ is the Turing Machine encoding, and $x_i[0]$ its binary input. The goal for each agent will be to print halted or not-halted. Optimization is not used.

0. $t=0$, initialization phase *init* : The definition of the i -th agent with its input - pair (M_i, x_i) , and the initial $\$$ -expression being parallel composition of agent processes. Parameters for the $k\Omega$ are selected: $k = b = 0$, $n = \infty$, Ω and $\$$ are irrelevant here. The goal is the minimum of the cost function (the default goal) is overwritten by the “do forever” goal in the infinite loop. Although it seems to be redundant here, technically speaking, we still use the $k\Omega$ -optimization with empty select and examine phases.

To start everything we run all processes in parallel

(|| $A_1 A_2 \dots$)

1. $t \geq 1$, first and other loop iterations, *sel*, *exam*, *exec* reduced to *exec* only

execute phase exec : for each instance of the Turing Machine and its input, the agent observes whether TM halted. If so it prints the result: “halted”. Otherwise it increments t and goes to the next loop iteration. We can think that the $k\Omega$ -search will stay a finite quantum of time in select/examine/execute iteration, and if not successful with reaching a decision will go to the next iteration, giving a next quantum of time. This can take either a finite amount of time/iterations if the TM halted reaching or not reaching its goal, or if the program loops forever, then the agent cannot decide in finite time whether the TM will halt or not. However, the agent by the infinity principle can wait “forever”, and in the infinity (in the limit) it will be able to decide and print: halted or not.

Two other proofs using the interaction and evolution principles are presented below.

Theorem 6: (On solution of the halting problem of UTM by the $\$$ -calculus using interaction principle) The halting problem for the Universal Turing Machine is solvable non-algorithmically by the $\$$ -calculus using the interaction principle.

Proof: There are two interacting agents, called UTM and Oracle, and both running the $k\Omega$ -optimization. The first agent UTM has as its input x , i.e., the encoding of the instance of the Turing machine with its input. The second agent Oracle has as its input a query from the first agent. No optimization is involved, i.e., $k = b = 0$, $n = \infty$, Ω and $\$$ irrelevant (no optimization) Everything is done in the *init* and *exec* phase. The default goal, minimum of the cost function is replaced by printing “halting” or “non-halting”.

0. $t=0$, initialization phase *init* : The two agents are defined (one encoding the instances of the Turing machine, and the second - an oracle), and the initial $\$$ -expression being parallel composition of two agents. Parameters for the $k\Omega$ are selected: $k = b = 0$, $n = \infty$, Ω and cost function $\$$ are

irrelevant (no optimization involved). The minimum of the cost function (the default goal) is overwritten by the “do forever” goal in the infinite loop. Although it seems be redundant here, technically speaking, we still use the $k\Omega$ -optimization with empty select and examine phases.

Both agent definitions:

$(:= \text{ (UTM } x) (\circ (\rightarrow a \ x) (\leftarrow a \ y) (\text{print } y) (\text{UTM new_x_instance})))$

$(:= \text{ (Oracle } x) (\circ (\leftarrow a \ x) (\rightarrow a \ (\text{select “halting“ “non_halting“})) (\text{Oracle } x)))$

Only the Oracle agent “knows” whether to select “halting” or “non-halting”, and the UTM agent simply promptly prints an answer.

To start everything we run both agent processes in parallel

$(\parallel (\text{UTM first_x_instance}) (\text{Oracle } x))$

1. $t = 1$, first loop iteration lasting forever; *sel*, *exam*, *exec* reduced to *exec* only

examine phase *exec* : successive interactions of UTM agent and Oracle agent are executed with interaction done by classical process algebra message passing involving pairs send and receive. UTM delegates the solution that is unable to solve itself, to a smarter Oracle. Practically, from the $k\Omega$ -optimization the inference rules for the execution of actions are needed only. Because $n = \infty$ and recursion is unconditional, the goal never will be checked again. The whole magic of Oracle is hidden in a non-recursive select function that optional definition is undefined, represented in the $\$$ -calculus as an atomic simple $\$$ -expression, defined as the “black-box”. We do not have an access to this black-box and its internal structure, but the Oracle agent somehow knows and can be queried about any Turing machine instance, giving an infallible answer. In infinity, all instances of TMs can be inspected.

Note that the readers may feel to be disappointed by such nonalgorithmic “proof” with a non-constructive “select” that nobody knows how to implement. However, the $\$$ -calculus allows to define formally simple $\$$ -expressions as arbitrary functions (including nonrecursive ones), thus technically this is correct, hiding the lack of algorithmic implementability by the atomic black-box select with an unknown structure.

Theorem 7: (On solution of the halting problem of UTM by the $\$$ -calculus using evolution principle) The halting problem for the Universal Turing Machine is solvable non-algorithmically by the $\$$ -calculus using the evolution principle. Proof: The idea is quite simple. The UTM (input to the $k\Omega$ -optimization) will be evolved in successive iterations (e.g., using the $\$$ -calculus send operator \rightarrow) to UTM with oracle, or to the infinite number of agents representing enumerable instances of TMs. Thus the halting problem using evolution principle will be transformed to the halting problem using interaction or infinity principles.

Assuming that a TM with an oracle can be encoded as a nonrecursive function ($\$$ -expression) using a binary alphabet and an ordinary TM can be (of course) encoded as a binary

string, thus a simple binary mutation changing one binary input to another can convert by chance an ordinary Turing Machine to the Turing Machine with an oracle [24]. The probability of such event is very small, and because nobody has implemented the Turing Machine with an oracle (although we know what its transition function may look like - see, e.g. [14]). We may have the big problem with the recognition of this encoding, thus even if theoretically possible, so far nobody has been able to detect the potential existence of the Turing Machine with an oracle.

D. Approximating the Universal Algorithm

We know from the theory of computation and the undecidability result from Section 5 that the search for the universal algorithm is a futile effort. However, is this truly so? In this section, we will try to provide some hope, but for the price of approximation of the best search algorithm.

We will show how the $\$$ -calculus can help potentially for the solution of the best search algorithm by approximating it. The best search algorithm will be in the sense of the optimum: finding the best-quality solutions/search algorithms for the all possible problems. The trouble and undecidability is caused by the requirement to cover exactly *all* possible problems. The number of possible algorithms is enumerable, however, the number of possible problems is infinite, but not enumerable (problems/languages are all possible subsets of all algorithms), see, e.g., [12].

Theorem 8: (On approximating the universal algorithm) The search for the universal algorithm is approximated in the limit by the $k\Omega$ -optimization if the search is complete an elitist selection strategy is used.

Proof: Let’s assume that the $k\Omega$ -optimization will have as its input a candidate for the universal search algorithm operating on its own inputs/problems to solve. In other words, the minimum of $\$_3(x[t])$ is looked for (the optimization problem) The solution will improve monotonically by elitist selection strategy. However, it will require an infinite number of iterations of the $k\Omega$ -optimization, checking a growing number of inputs for each universal algorithm candidate instance, but this is consistent with the infinity principle. By completeness, the probability of finding each universal algorithm candidate is greater than zero and each (reachable) search algorithm is guaranteed to be inspected by an exhaustive search in infinity. Assuming that the number of possible search algorithms is infinite but enumerable (a reasonable assumption, because all algorithms/TMs can be encoded over, for instance, a binary alphabet and enumerated), the best search (universal) algorithm (a nonexistent, or more precisely, an asymptotic optimum) will be approximated by the $\$$ -calculus after an infinite number of iterations to the arbitrary degree of precision. However, the optimum (the universal algorithm) has to be outside of the search tree, otherwise by completeness it would be reached in the infinity and problem would be decidable (what is not possible according to Theorem 3).

If the reader may question the practicality of the above “infinite recipe” to find the universal algorithm, we should

say that at least it provides a hope that the attempt to look for the universal algorithm is not so futile, and we will improve continuously the currently known “best” universal algorithm never reaching it.

Note that it is not true that the theorem is “impractical”. Of course, we are not guaranteed to explore in the finite time the whole infinite search space of potential algorithms. However, the above theorem gives a quite practical clue stating that the search for the universal algorithm is not so hopeless. We can improve indefinitely the quality (i.e., the generality) of search algorithms. However, we do not guarantee that the best algorithm will be found in the finite or infinite time (because the problem is TM/\$-calculus undecidable). All that can be claimed is that it can be indefinitely approximated.

It is quite different to claim that yes, you can improve your solution for the price that you will never reach an optimum in the final or infinite time, compared to a flat negative statement: “do not look for the best algorithm because such effort is fruitless”. The theorem says: look for the best algorithm; you are not guaranteed to find it in the finite or infinite time, but as a side effect of your search you will get (hopefully) better and better search algorithms. In such a way, for instance, the development of the whole area of mathematics, works. Finding an axiomatic description of all theorems in mathematics is impossible (the Entscheidungsproblem - Hilbert’s decision problem is undecidable [23], [12], [7], [9]). However, in spite of that, new generations of mathematicians and computer scientists work and generate new theorems and improve indefinitely the current state of art of mathematics and computer science. The famous Entscheidungsproblem does not prevent us from discovering new theorems, improving our knowledge of mathematics, but we will never be able to write all theorems.

VI. Conclusions

In the paper, we investigated expressiveness of the π -calculus and the $\$$ -calculus. We justified that both process algebras belong to superTuring models of computation, i.e., are more expressive than Turing Machines.

To gain higher expressiveness, superTuring models employ the interaction, evolution and infinity principles. It is unclear at this moment, how combination of several principles may increase expressiveness of such models. We know that the application of one principle only is sufficient to derive models more expressive than TMs. Very little is known about the relations between superTuring models. Some preliminary results are known that random automata networks are at least equally expressive as discrete neural networks, and discrete neural networks are at least equally expressive as cellular automata. We know also that Sequential Interaction Machines are less expressive than Multistream Interaction Machines, and that the $\$$ -calculus can simulate the π -calculus and random automata networks, but much more research is needed.

Acknowledgments

Research partially supported by the grant from Office of Naval Research ONR N00014-06-1-0354.

REFERENCES

- [1] Bergstra J.A., Ponse A., Smolka S.A. (eds.), Handbook of Process Algebra, North Holland, Elsevier, 2001.
- [2] Church A., An Unsolvable Problem of Elementary Number Theory, American Journal of Mathematics, vol.58, 1936, 345-363.
- [3] Church A., The Calculi of Lambda Conversion, Princeton, N.J., Princeton University Press, 1941.
- [4] Eberbach E., A Generic Tool for Distributed AI with Matching as Message Passing, Proc. of the Ninth IEEE Intern. Conf. on Tools with Artificial Intelligence TAI’97, Newport Beach, California, 1997, 11-18.
- [5] 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.
- [6] 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.
- [7] 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, Modelling and Algorithms, Narosa Publishing House, New Delhi, 2003, 1-32.
- [8] Eberbach E., Wegner P., Beyond Turing Machines, The Bulletin of the European Association for Theoretical Computer Science (EATCS Bulletin), 81, Oct. 2003, 279-304.
- [9] Eberbach E., Goldin D., Wegner P., Turing’s Ideas and Models of Computation, in: (ed. Ch.Teuscher) Alan Turing: Life and Legacy of a Great Thinker, Springer-Verlag, 2004, 159-194.
- [10] Eberbach E., $\$$ -Calculus of Bounded Rational Agents: Flexible Optimization as Search under Bounded Resources in Interactive Systems, Fundamenta Informaticae, vol.68, no.1-2, 2005, 47-102.
- [11] Hoare C.A.R., Communicating Sequential Processes, Prentice-Hall, 1985.
- [12] Hopcroft J.E., Motwani R., Ullman J.D., Introduction to Automata Theory, Languages, and Computation, 2nd edition, Addison-Wesley, 2001.;
- [13] Horvitz E., Zilberstein S. (eds.), Computational Tradeoffs under Bounded Resources, Artificial Intelligence 126, 2001, 1-196.
- [14] Kozen D.C., Automata and Computability, Springer-Verlag, 1997.
- [15] Milner R., A Calculus of Communicating Systems, Lect. Notes in Computer Science vol.94, Springer-Verlag, 1980.
- [16] Milner R., Parrow J., Walker D., A Calculus of Mobile Processes, Rep. ECS-LFCS-89-85 and -86, Lab. for Foundations of Computer Science, Computer Science Dept., Edinburgh Univ., 1989.
- [17] Milner R., Parrow J., Walker D., A Calculus of Mobile Processes, I & II, Information and Computation 100, 1992, 1-77.
- [18] Milner R., Communicating and Mobile Systems: The π -calculus, Cambridge University Press, 1999.
- [19] Parrow J., An Introduction to the π -calculus, in: (ed. J.A. Bergstra) Handbook of Process Algebra, Elsevier, 2001, 479-543.
- [20] Plotkin G., Stirling C., Tofte M. (eds.), Proof, Language, and Interaction: Essays in Honour of Robin Milner, The MIT Press, 2000.
- [21] Post E., A Variant of a Recursively Unsolvable Problem, Bulletin of the AMS 52, 1946, 264-268.
- [22] Sangiorgi D., Walker D., The π -calculus: A Theory of Mobile Processes, Cambridge University Press, 2001.
- [23] 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.
- [24] Turing A., Systems of Logic based on Ordinals, Proc. London Math. Soc. Series 2, 45, 1939, 161-228.
- [25] Wegner P., Eberbach E., New Models of Computation, The Computer Journal, 47(1), The British Computer Society, Oxford University Press, 2004, 4-9.
- [26] 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.