Interaction, Evolution, and Intelligence

Dina Goldin
Department of Mathematics and Computer Science,
University of Massachusetts at Boston,
Boston, MA 02215
dqg@cs.umb.edu

David Keil
Department of Mathematics and Computer Science,
University of Massachusetts at Boston,
Boston, MA 02215
dkeil@cs.umb.ed

Abstract- Evolutionary computation (EC) and intelligence are closely related. Adaptability can be considered an indispensable aspect of intelligence; in turn, interaction of an agent with its environment is necessary for adaptation.

Interaction is fundamental to EC at multiple levels, and an interactive viewpoint enhances our understanding of EC principles. Interaction is based on a stream-based rather than string-based view of computation, modeled mathematically by coinduction and coalgebras. It forces us to rethink some fundamental assumptions of computer science.

The challenge of building intelligent systems can be better met by accepting a paradigm shift from traditional algorithmic models of computation toward interaction, with all its implications.

1 Introduction

1.1 Evolution of Computing

The word "computer" originated as a job category for persons who repeatedly executed tedious mathematical calculations over sets of data to obtain desired answers. The original electromechanical computing machines carried out the same tasks automatically, playing a significant role during World War II.

Computer science has been evolving very rapidly, and present-day uses of computing are very different from its origins. Computing devices provide a wide range of services that have become essential to our society and our daily life. Software systems are becoming more powerful yet friendlier to naïve users.

The increased attention to *agent-based computing* is part of the evolution, and puts into perspective the related themes of *interaction*, *evolution*, and *intelligence*. Piaget associated intelligence with an evolved ability to adapt to one's environment [Pia50]. A similar view is held in EC literature [Fog00]. If computer systems are tending toward greater ability to emulate human intelligence, it is because of their adaptability and interactiveness, not merely their raw processing power.

1.2 The paradigm shift towards interaction

There is continuous interaction between today's computing systems and their environment, e.g., a word processor and a human, a control system and a physical process being controlled, embedded systems and their sensors/actuators [Es00], as well as concurrent interactions over networks. Rather than just perform individual transformations of input data to output data, today's computers deliver a continuous *service* to their clients, executing tasks that involve multiple inputs and outputs throughout the lifetime of the computation [Weg97]. We are seeing a spread of reactive and adaptive systems, e.g. in electronic devices, robots, and process control.

This is a paradigm shift, inviting us to rethink the fundamental notions of computatation, computational models, and *computational problems*. Today's computational problems imply processes or *policies* rather than just computational paths from an input to an output. Our models of computation must be enriched to model *persistence of agent state*, infinite *interaction streams*, and ongoing interaction with a (possibly uncomputable) *environment* [WG99a].

The Turing Machine is a formal model of *algorithmic* computation. Algorithmic computing systems accept input from the environment and then "shut out the world" as they compute the output. The Church-Turing thesis codified the observation that the intuitive notion of an effective procedure (algorithm) is embodied in the theoretical model of the Turing machine or, equivalently, the Lambda calculus [Tu36].

There is no accepted canonical model of interactive computation to date, but there is much recent work in this direction [Weg98, GS01, VW00]. For example, [VW00] argues that "the underlying classical Turing paradigm has to be changed," urging an extension of the Church-Turing thesis to encompass interactive computation.

1.3 Interaction and EC

Ongoing interactions of three kinds occur in EC:

(a) *Adaptation* implies the interaction of a computing agent, such as a robot or softbot, with its environment;

(b) *Competition* implies interaction among multiple agents, such as offspring in evolutionary programming (which shapes the evolution of two genetic pools of self-reproducing agents [BH97]);

(c) *Evolution* implies interaction between a multi-agent system (the evolving population) and its environment; this may be at a higher level than the interactions of (a).

Likewise three levels of *learning* may be distinguished in EC:

ontogenetic (adaptation of an individual), *sociogenetic* (adaptation by a population), and *phylogenetic* (arising from a lineage of generations) [Fo00]. Learning is an example of *history-dependent behavior* that cannot be modeled within an algorithmic setting, where computations have no history.

Using living systems rather than mathematical functions as their model, EC researchers have for some forty years discussed automated systems that adapt interactively to their environment and that may evolve "in fast time," to the extent of being able to "predict the future" based on percepts obtained from their environment [Hol62, Fo62].

Learning is part of adaptation. Contrary to supervised learning or learning by memorization, which may be modeled by reading a file before executing an algorithm, adaptation modifies system response to future inputs as a consequence of past input.

The evolution of natural species is in part an evolution of the individual *behaviors*. The field of evolutionary programming works to codify evolved behaviors as executable programs and modules. The fundamental idea of applying evolutionary principles to generate computer programs can be traced back to Turing himself, who suggested that intelligent (adaptive) machines might evolve under the artificial-selection regime of the experimenter. However, the evolutionary process he envisioned was not automatic [Tu50].

Related closely to evolutionary computing is the phenomenon of *emergent behavior* of *complex* systems [Sim70], multi-agent interactive systems such as human organizations or intelligent software systems. It refers to complex adaptive (evolving) systems whose behavior is more than the sum of the behaviors of the components (agents).

Outline. We begin by describing some concepts developed to formalize interactive models of computation (Section 2), and describe how interaction is part of intelligence (Sections 3 and 4). A mathematics that goes beyond induction is needed to model the stream-based behavior found in interactive computing and in EC (Section 5). A study of multi-stream interaction reveals that parallel nonserializable interactions among agents are more powerful than either transformations of data or sequential interactions (Sec. 6).

2 Interactive models of computation

In this section, we present some of the formal notions for modeling interactive computation that are used in later sections. Formal models of interactive computation include the pi calculus and calculus of communicating systems [Mil80, Mil99], input/output automata [LT89], and abstract state machines [Gu93]. By contrast, our model will be based on Turing Machines (TMs), extending them to obtain a simple model of computation that is interactive and capable of modeling adaptive, history-dependent behavior. An equivalent model based on interactive transition systems is also possible [GS00].

2.1 Turing Machines and algorithmic computation

TMs model computations from strings to strings ($\Sigma^* \rightarrow \Sigma^*$), or equivalently from natural numbers to natural numbers ($\mathbf{N} \rightarrow \mathbf{N}$). Any computation of a TM is a sequence of state transitions that always starts in the TM's *initial* state and may end in a *halting* state. The transitions are specified by a transition function, which is fixed for a given TM. Given the same input string, two computations of the same (deterministic) TM will be identical.

TM computation is *algorithmic*, where the values of all inputs are predeterminted at the start. It cannot be affected by subsequent changes to the environment; therefore, TMs compute as *closed* systems. Even though it is commonly throught that the TM computations are "what computers do", they actually model only individual input-processing-output steps of a computer, i.e., batch computing, or, in a GUI environment, a single event and the system's response.

Whereas a Turing machine stores no record of, and has no use for, previous input/output, this is not the case for adaptive agents. Both they and their environment can change as a result of interaction; the agent's behavior is shaped by this interaction history. History dependent behavior is characteristic of *interactive* computation, but absent from algorithmic computation.

2.2 The characteristics of interactive computation

Interactive computation, such as that of adaptive agents, is characterized by three features absent from algorithmic computation: (a) interleaving of inputs and outputs during computation; (b) stream input/output; (c) history-dependent behavior of the computing agent. Having looked at the latter in Section 2.1, we consider the other two here.

Interactive computation involves the exchange of finite messages between the computing agent and its *environment*, where the agent's response to earlier input messages can affect the contents of later input. The *environment* of an agent is the origin of the agent's inputs and the destination of its outputs. In an interactive computation, the environment consists of the set of entities or agents with which it exchanges messages. Interaction with these agents may be direct or *indirect*, where the messages pass through a *mediator in the*

environment, e.g. ants depositing or sensing pheromones to communicate with other ants.

Interactive computation may infinite. In the latter case, the agents may be carrying out a *service* or executing a *process*; these forms of computation are inherently interactive. Infinite interactive computation is modeled by *streams* of input and output tokens. A *stream* over *S* is an infinite sequence of values (tokens) drawn from a countable set *S*. The infinite nature of streams corresponds to the (possibly) infinite nature of interactive computation, such as evolutionary computing or a control process, in contrast to the finite, terminating nature of algorithmic computations. Section 5 discusses the coinductive foundations of this infinite nature of streams.

Furthermore, an environment of a computation may be uncomputable, such as "the weather" [Mil80]. System designers today model the environment by use cases and actors (UML). Software engineering and object-oriented programming model it by input/output streams, by messages, and by event loops.

2.3 Persistent Turing Machines

A model of interactive computation can be obtained with a minimal extention to the TM model; we call it a *persistent Turing machine* (PTM). PTMs are 3-tape TMs, where the tapes are for *input*, *internal work*, and *output* respectively. A single computational step (*macrostep*) of this machine consists of a halting TM computation, where the original contents of the tapes is (w_{in}, w_{mem}, ε) and the final contents is (w_{in}, w'_{mem}, w_{out}), respectively. Hence, a PTM defines a partial recursive function from pairs of strings (w_{in}, w_{mem}) to pairs (w_{out}, w'_{mem}).

Given an input stream $\{i_1,i_2,i_3,\ldots\}$, a computation of a PTM *M* proceeds by macrosteps, processing the input tokens one at a time while generating an output stream $\{o_1,o_2,o_3,\ldots\}$. *M*'s work tape will also *evolve* during this computation; we refer to its contents as *M*'s *memory*:

$(i_1, \varepsilon) \rightarrow (o_1, m_1)$
$(i_2, m_1) \rightarrow (o_2, m_2)$
$(i_3, m_2) \rightarrow (o_3, m_3)$
…

A *sequential interaction machine* (SIM) is an interactive computing device with a single *interaction stream* [WG99a]. PTMs formalize the SIM notion. For example, the interaction stream for the above computation is:

$\{(i_1, o_1), (i_2, o_2), (i_3, o_3), \ldots\}$

Note that only the inputs and outputs are *observable* by the environment, and part of the interaction stream. The memory, which is *persistent* (its value is preserved from the end of each macrostep to the beginning of the next), is internal to the PTM.

To take a simple example, let an *answering machine A* be a PTM whose work tape contains a sequence of recorded messages. If A allows the operations *record*, *playback*, and *erase*, then its Turing-computable function might be:

f_A (*record Y, X*) = (*ok, XY*);
f_A (*playback, X*) = (*X, X*);
f_A (*erase, X*) = (*done,* ε).

where ε is the empty string.

As an example computation of *A*, the input stream (*record A, erase, record BC, record D, playback*, …) generates the output stream (*ok, done, ok, ok, BCD*, …); the state evolves as follows: (ε, *A*, ε, *BC, BCD, BCD*, …).

This small example suffices to illustrate the three characteristics of interactive computation presented in Section 2.2. In particular, the history dependent behavior is reflected in the output of the fifth macrostep, which depends on inputs from steps 3 and 4.

2.4 Stream languages and PTM expressiveness

The observable behavior of PTMs is formalized with *stream languages.* Where *M* is a PTM, *M*'s *persistent stream language PSL*(*M*), is the set of interaction streams observable on *M*, such as $\{(i_1,o_1), (i_2,o_2), (i_3,o_3), \ldots\}$ above.

We can also consider what happens if the memory is not persistent, and the worktape is reinitialized at each step. For the input stream (i_1, i_2, i_3, …), the computation would be as follows:

$(i_1, \varepsilon) \rightarrow (o_1, m_1)$
$(i_2, \varepsilon) \rightarrow (o_2, m_2)$
$(i_3, \varepsilon) \rightarrow (o_3, m_3)$
…

The *amnesic stream language ASL*(*M*) is the set of streams observable on *M* in this new context. We call a PTM *M amnesic* if PSL(*M*) = ASL(*M'*), for some *M'*. Essentially, a PTM is amnesic if it either erases the contents of its worktape at the beginning of each macrostep, or just ignores it throughout the macrostep. An example of an amnesic PTM is an agent that processes queries over a fixed database, or solves an algorithmic problem for each input.

Though amnesic PTMs represent a more natural extension of TMs to a stream-based setting, persistence of memory is crucial for PTM expressiveness:

Theorem: The set of ASLs is strictly contained in the set of PSLs [GS00].

This theorem also gives a formal basis to the claim that simple reflex agents are not as powerful as state-based agents [RN95].

Any discrete-state computing agent that interacts sequentially with its environment may be modeled by a PTM. For example, a person driving a car may be so modeled (see Section 3.2). A reactive program, such as one that results from an evolutionary-programming process may be modeled by a PTM; in fact, a process of evolutionary

computation itself may be modeled as a PTM but not as a TM.

3 Interaction and intelligence

The sequential interactive model of computation allows us to define stronger criteria for intelligence than is possible with an algorithmic model of computing. These criteria reflect the system's ability to learn and adapt its behavior to its environment on the basis of its interaction history [WG99c].

Whenever two agents are computing within different environments, or embedded in different locations, their interaction histories naturally differ. Adaptive agents that start out as identical may adapt differently if their interaction histories differ. Then, they will no longer be identical, and will return different output for the same input. For example, a driving agent that learns to lower its speed after getting speeding tickets will behave differently from an identical agent who learned to drive in a police-free environment. Such adaptation to the environment is an example of history-dependent behavior.

By contrast, for a given input string, (deterministic) TMs always return the same output string; their behavior cannot be history dependent. It follows from the Church-Turing thesis that algorithmic computation, for which TM is a canonical model, is not expressive enough to model adaptation or evolution.

3.1 Intelligence: question-answering or adaptation?

Though it is hard to define intelligence in a way that everyone can agree on, it is easier to agree on when one system is "smarter" than another. Smart bombs are smarter than other bombs because they can adjust their course in mid-flight using feedback from sensors. Smart interviewees for jobs will try to project more confidence if told in the interview that the position requires someone with confidence. When a task is the same, the system exhibiting a greater ability to adapt to the environment will exhibit greater intelligence.

The first definition of computational intelligence was proposed by Turing, who approached it behavioristically and anthropomorphically. Turing's definition of computing by a machine was the starting point of his definition of intelligence in terms of a precise notion of computing. The *Turing test*, now part of AI lore, stipulates that a system is ultimately intelligent if it cannot be distinguished from a human through question-answering [Tu50]. This test is not interactive: later questions are worded independently of answers to earlier ones.

It is easy to see that a non-interactive framework will not give the best possible test of intelligence. It cannot test the agent's ability to learn or to anticipate the questioner's needs – in sum, to adapt. It is not able to *distinguish* between an *amnesic* agent and a *persistent* one (Section 2). Thus, this test's conceptualization of intelligence is incomplete, not measuring ability to adapt, sometimes referred to as "street smarts."

For a concrete example where a non-interactive test is provably weaker, consider an investigator (*questioner*) who is interrogating a suspect (*answerer*), in an attempt to establish guilt. He can ask the suspect to fill out a questionnaire, or he can conduct an interactive interrogation with follow-up questions. Clearly, anyone who can give convincing answers to an interrogator in person can do the same on the questionnaire. But the reverse is not necessarily true. It is possible to imagine cases where suspects can fill out any questionnaire without implicating themselves, but an interactive interrogation can exploit weaknesses in their story to lead to inconsistencies that establish their guilt.

The reason why a suspect may fail an interrogation despite being able to pass all questionnaires is that interactive questioning forces the answerer to *commit* to earlier answers before seeing later questions; it also allows the questioner to base later questions on the answers to earlier ones, to probe for weaknesses.

When comparing two tests, if one can make finer distinctions (e.g. of intelligence, or of guilt) than the other, we consider the first more powerful, or even more intelligent. As the example above shows, interactive testing is more powerful than algorithmic testing. The extension of the notion of computation to include interaction allows us to define a more powerful form of intelligence by extending the Turing test.

Behavior-based views of intelligence are common nowadays, serving as a foundation for "new AI". This behavior-based approach to AI uses *rationality* as a criterion of intelligence, in the sense of ability to "do the right thing" [RN]. The interactive Turing test allows us to reconcile such behavior-based views of evaluating intelligence with the adaptation-based view of intelligence found in EC literature [Fo00].

3.2 Rethinking computability

Having reconsidered our criteria for intelligence, we also need to reconsider what kind of *problem* requires intelligence to solve. Consider d*riving from point A to point B*. The solution to this problem is in the form of be a series of actions, such as pressing the accelerator or turning the wheel. In a "toy" world described by some map, this seems a simple task. But the presence of traffic and weather conditions, as well as potholes and stones, can affect the car's motion. The problem of driving under these conditions cannot be reduced to an

algorithm even in principle: it depends on incredibly complex unpredictable events that are not algorithmically describable [Weg98].

Though driving in traffic is not solvable when approached algorithmically, there are already agents being built that can accomplish this task (solve this problem). They do so interactively, with continuous feedback from a video camera. Thus the problem of driving is an interactive one, approached as a sequence of algorithmic subproblems, analyzing one video frame at a time and deciding in real time on the proper next action. These algorithmic subproblems are tractable, rendering the overall interactive task tractable as well.

The driving example shows the difference between algorithmic and interactive notions of computability. Only problems whose input can be completely specified a priori are computable algorithmically. Algorithmic notions of computability do not take the environment into account, essentially assuming a single absolute environment.

By contrast, interactive computability of a task is conditional on the *environment* of the agent. The agent is providing a computational service to its environment; whether or not the service can be delivered depends on the agent's environment as much as on the definition of the task.

Assumptions about the environment (e.g. *constraints* on it) are part of the interactive problem specification, determining the problem's feasibility. For designing interrogators, we assume they will be talking to humans and not animals or aliens. For designing cars, we assume they will be driving on a road, on Earth; we may even assume a paved road not covered by snow.

The intrusion of the environment into the notion of an interactive problem causes us to reconsider the notion of *feasibility* for such problems. What is feasible in some environments is not so in others. For example, the problem of running a certain Java applet from a web server may be considered "unfeasible" if it loads so slowly with the current technology that the users will not be able to wait for it.

3.3 Efficiency of the evolutionary process

Due to the presence of interaction, evolution can be exponentially more efficient than exhaustive search. Let us consider the problem of creating agents. To choose even an eight-symbol, ten-state finite automaton design by an exhaustive search, one must select from a space of 10^{152} possibilities [Fo00], due to a combinatorial explosion in the number of possible machine designs of even trivial size. Even billions of years, with a parallelism factor of billions, would not be enough to produce something as complex as a human.

Clearly something else is going on with evolution, besides an algorithmic search. Another example where nature is astonishingly fast, though at a different level of magnitude, is *protein folding*. The fastest machines available take hours if not days of computational time to do something that our cells do in an instant.

In both cases, the explanation is that the corresponding real-world process is non-algorithmic. The interactive nature of this process allows it to influence its environment, which in turn influences the system to "help" it with the computation. For example, once the protein folding starts, pulled in some particular direction by the initial forces, there is a continuous adjustment of the forces during the folding, until the equilibrium is reached and the protein is in the final position.

Evolution must be an *anytime* process. An interruptible *anytime algorithm* "evolves" its solution rather than producing it at once; the later the algorithm is terminated, the better the solution [Eb00]. Anytime algorithms are suitable for the computational steps in interactive systems. When the computational steps are "anytime," the interactive system becomes more flexible, enhancing its problem-solving ability.

4 Evolution, learning, and interaction

It has been pointed out that natural evolution of organisms can be seen as "a very long term form of learning" [Br89]. Learning is an adaptive process of sharpening the ability to predict future events and their probabilities, including conditional prediction [Fo00]. Equipped with well-honed predictions, the agent is better prepared to survive or to achieve other goals.

4.1 Adaptive agents and their environments

As the agent lives (computes), its *environment* consists both of other agents, with which it interacts, and the environment shared with such agents. This is true in natural as well as artificial systems [BH97], where agents may be called *components* or *objects*.

The universal purpose of learning is to adapt to one's environment by predicting future inputs or percepts and thus to be able to generate appropriate outputs or actions. In an inaccessible, continuous, dynamic, non-episodic environment, it is impossible for the agent to build or deduce a complete picture of its world and the probabilities that characterize this world. Hence an adaptive system must carve out a set of estimated *constraints* on probabilities of seeing a given input at a given time.

In *reinforcement learning*, the emphasis is on hypothesis formation, verification, and

correction, using interaction throughout. This continuous hypothesis testing by observing the effect of one's actions upon the environment prefigures, on a lower scale, the scientific method, which is also clearly interactive. Natural systems have evolved reinforcement learning; EC has not yet done so to a significant extent.

As a cognitive model of the environment evolves, it is honed to correct rules or to cover exceptions to earlier rules [HH89]. This is similar to the process of observing a system: further observations permit the observer to make finer and finer distinctions about the system. But the internal structure of the system, and even its total behavior, cannot be guaranteed to be fully determined though finite observation [Weg98].

4.2 Evolution as intelligence

Jean Piaget pointed out that intelligence is a particular instance of evolved adaptation in nature, or "a form of equilibrium towards which the successive adaptations and exchanges between the organism and his environment are directed" [Pia]. Intelligence is a feature of a system, not of a particular algorithm [Fo00].

Adaptation requires prediction, control, and feedback. Since the environment may change, the output function computed at each step of an adaptive computation process may change. The process is non-algorithmic, both in its theoretically infinite duration and in its history-dependent features.

Evolution's use of reproduction, variation, competition, and selection provides an inspiration for a method creating intelligent artificial systems. Design variations acquired by such systems is passed on to their offspring [Fo00]. As a process to optimize solutions to specific problems [BH97], evolution is clearly nonterminating and hence nonalgorithmic as long as an environment changes dynamically.

The general process of evolution of a species cannot be optimized. Optimality depends strictly upon problem specification, and the formulation of interactive problems is dependent on the environment. The no-free-lunch theorem [Fo00] states that no evolutionary algorithm can excel for all problems. The interaction between an EC system and its environment determines the best method for building adaptive agents. Various kinds of intelligence are shaped by different environments, and no solution will work in all environments.

5 Interaction, streams, and coinduction

The environment where learning and evolution occur is observable as streams of inputs and outputs. Whereas algorithms have finite *string* input, interactive computation is associated with infinite *streams* of inputs. Living and artificial agents interact with their environments through streams of percepts and actions.

The unending lineage of future descendants of an agent or organism is another stream, this one generated by the evolutionary process. Since a population has many components and many lineages, we must consider multiple infinite streams. The infinite character of evolution was noted many years ago by researchers in EC [Hol87].

New mathematical tools are needed to model stream-based computation, because inductive methods of definition and reasoning only work in domains of finite objects. [WG99a]. The chief new notions are *coinduction*, *coalgebras*, and *non-well-founded sets* [JR97, BM96].

Inductive definitions provide three conditions: (1) initiality, (2) iteration, and (3) minimality. For example, the set Σ^* of all strings over an alphabet Σ is inductively defined as follows: (1) the empty string $\varepsilon \in \Sigma^*$; (2) if $a \in \Sigma$ and $x \in \Sigma^*$, then $ax \in \Sigma^*$; (3) Σ^* contains no other elements. The equivalent expressiveness of TMs, algorithms, computable functions, and formal systems is due largely to their common use of induction to define static domains and dynamic processes of derivation.

While induction formalizes the metaphor of constructing finite structures from primitives, coinduction formalizes the *observation* metaphor of stream-based environments. *Coinductive* definitions eliminate the initiality condition of induction, and replace the minimality condition by a maximality condition.

For example, the set of streams over a set of tokens S is defined coinductively as follows. *Iteration condition:* If t is a stream over an alphabet S, and $a \in S$, then (a,t) is a stream over S, consisting of the token a followed by a stream t. *Maximality condition:* the set of streams over S is the maximal set satisfying the iteration condition:

$$S^{\infty} = \{(a,t) \mid a \in S, t \in S^{\infty}\}.$$

Note that this definition seems circular, without a base case. Coinduction replaces *the least fixpoint semantics* associated with inductive formal systems by *greatest fixpoint semantics* [WG99b]; as a result, this definition is well-formed. The Anti-Foundation Axiom, of non-well-founded set theory, states that flat set equations, including circular ones like that for Σ^{∞} above, have unique solutions [BM96].

Coinduction provides a mathematical framework for formalizing systems that interact with the external world though infinite interaction sequences. In addition to greatest fixpoints, the semantics of coinduction assumes *lazy evaluation*; the tokens of the stream are observed one at a time, rather than all at once. Hence,

coinductive definitions permit us to consider the space of all processes as a well-defined set, even if the input streams are generated dynamically and cannot be predicted a priori.

6 EC and multiagent interaction

Until this point we have been discussing sequential interaction based on single I/O streams. The strength of natural evolution includes not only the interaction of individual organisms with their environment, but also the massive concurrency of these interactions, where the environment is shared. Certain kinds of fitness testing are an example of concurrent interaction—battle in competition for mates and food, for example. Models for such interaction must be part of the theoretical foundation of EC.

6.1 Concurrency and multi-stream interaction

Database and operating-systems theory prescribes handling the difficult aspects of concurrent and distributed computing by use of the notion of *atomic* (indivisible) transactions and *serializable processes*, reducible to a sequence of atomic transactions. But in nature and in arbitrary computing environments, serializability is not guaranteed. A pack of predators does not wait in a line, for example. Thus we must account for a kind of interaction that sequential machines cannot model.

A *multi-stream interaction machine* (MIM) is a state-transition system that interacts with autonomous multiple streams [WG99a, WG99b]. Examples of multi-stream interaction are ant swarms, the Internet, traffic patterns, and economic markets.

Multi-agent interactive behavior is part of a taxonomy of computational phenomena of which algorithms are only one element (Figure 1).

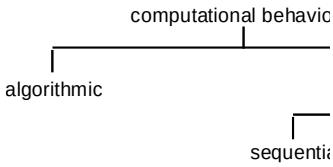

Figure 1: *Classifications of behavior by expressiveness*

Co-evolution is an example of the power of parallel and concurrent (multi-stream) interaction. In co-evolution, the products of the evolutionary process define their own fitness function, because they are selected on the basis of their success in competing with each other [FW93]. An example is the evolution of a predator species in nature alongside that of a prey species.

6.2 Emergent behavior in multi-agent computing

Not only do multiple creatures (or agents) interact in a natural (or artificial) environment; an agent may be composed of multiple interacting components. Walking, for example, is a behavior that can occur in robots or insects with little centralized control, using simple reflexes localized in the limbs [Br89].

The behaviors of systems or societies of agents can well be more complex than the structures of the agents themselves; *emergent behaviors* of a system are more than the sums of the behaviors of the system's parts.

This noncompositionality may be seen in simulations such as the StarLogo [SL00] termite simulation. The termites have limited or no capacity for planning, coordination, and perception, but they have an evolutionary need to build circular piles of wood chips. Evolution has led termites to apply a simple interactive strategy: move at random, pick up a chip whenever an isolated one is encountered, and put it down once it bumps into more chips.

The result is as the certain as if one were to sweep together a pile of scattered wood chips by means of planning and coordination. But the termites accomplish the same task, creating order out of chaos, without even an internal representation of the goal. *Ant computing* and *swarm intelligence* are names for a branch of research related to self-organizing robotics based on similar phenomena from the natural world.

6.3 Evolutionary multi-agent systems

Once we accept the idea that interaction is more powerful than algorithms, we can distinguish multiple levels in the capability of systems to express or observe intelligent behavior. Mutli-agent computing is more expressive than the sequential kind because it can express phenomena such as collaboration or delegation. Collaboration and coordination of multiple evolving robots cannot be represented by a Turing machine, or even a sequential interaction machine, because the interactions may be non-serializable.

The greater expressiveness of multi-agent interaction over sequential interaction contrasts with the result that multi-tape Turing machines are no more expressive than single-tape ones. Beyond a single-agent Turing test (Section 3), a multi-agent Turing test may also be defined, observing the agent's interaction with multiple autonomous agents to evaluate its distributed intelligence.

Algorithmic, sequential, and distributed thinking are progressively stronger forms of behavioral approximation to human thinking, defined by progressively more stringent tests of behavior. Sequential thinking is realized by SIMs and used by observers who ask follow-up questions (make sequential observations). Distributed thinking is

realized by MIMs and requires multi-agent observation of system response to multiple potentially conflicting asynchronous requests [WG99a, WG99b].

No sequence of preordained steps, or even a series of interactions, can model the multiple-stream encounters between multi-interacting evolved agents and their environments. By creating, concurrently fitness testing, and pruning successive generations of candidate programs, an evolutionary programming environment is carrying out a process analogous to what distinguishes multi-stream from sequential interaction. It is more powerful for a manager to add to, remove from, and otherwise reorganize a work group than to simply assign tasks to the group. Likewise, an evolutionary-programming process makes use of higher functions than a programmer, using modules that generate and test code automatically, rather than doing it directly.

Research in various fields of EC has described multi-agent interaction, from embodied evolution using robotics to modeling social structures, economic phenomena, and military situations, as survey articles have noted [BH97]. [Min86] points out that the mind also represents multi-agent interaction; in the "society of mind", the "thinking" that occurs is really interactive neuronal behavior. The study of intelligent systems implies the study of multi-stream interaction.

7 Conclusion

Building intelligent systems has been an important goal in the evolutionary computing community. The history of the endeavor to create intelligence has had its failures, to be expected of something so ambitious. One of them was the Japanese-sponsored Fifth Generation Computing project, aimed at using logic programming to implement artificial intelligence. The failure of this project can be attributed to the limits of the context of closed systems and first-order logic, within which is was founded [Weg99].

Algorithmic computing confined within the limits of Turing machines is not expressive enough for the problem of intelligence. But natural evolution has solved this problem of intelligence, with humans as the existence proof. This is an encouragement to attempt to build artificial intelligence using systems that are interactive and evolutionary in character. Interactive systems can overcome the limits of the algorithmic approaches.

EC merges several hitherto disparate areas, such as genetic algorithms and evolutionary programming. Much work remains to be done to establish it as a unified field of research with a common foundation [BH97], akin to what has happened in AI with its recent focus on rational agents [RN95]. The paradigm shifts toward evolution and toward interaction can strengthen each other. We believe that a perspective that focuses on the interactions found in evolutionary computation can contribute significantly to establishing the unified foundations of this field.

Bibliography

[BH97] Thomas Bäck, Ulrich Hammel, and Hans-Paul Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions in Evolutionary Computation 1*(1) (April 1997), pp. 3-16.

[BM96] Jon Barwise and Lawrence Moss. *Vicious circles* (CSLI Lecture Notes #60). Stanford: CSLI Publications, 1996.

[Br89] Rodney A. Brooks. A robot that walks: Emergent behaviors from a carefully evolved network. MIT AI Lab A.I. Memo 1091, 2/89.

[Eb00] Eugene Eberbach. $-calculus intelligent agents. *http://dragon.acadiau.ca/~eberbach/projects/$-calculus.html.*

[Es00] D. Estrin, R. Govindan, and J. Heidemann, Guest Editors. Embedding the Internet (special multi-article section). *Communications of the ACM, 43*(5) (5/00).

[Fo00] David B. Fogel. *Evolutionary computation: Toward a new philosophy of machine intelligence*, 2nd edition. IEEE Press, 2000.

[Fo62] Lawrence J. Fogel. Autonomous automata. *Industrial Research* 4(2/62), pp. 14-19.

[FW93] Harald Freund and Robert Wolter. Evolution of bit strings II: A simple model of co-evolution, *Complex Systems 7* (1993), pp. 25-42.

[GS00] Dina Goldin; Scott A. Smolka; Peter Wegner. Turing machines, transition systems, and interaction. University of Massachusetts Boston CS-Math Tech. Rep. 00-07, Oct. 2000.

[Go99] Dina Goldin. Persistent Turing Machines as a model of interactive computation, Presented at FoIKS'00, Cottbus, Germany, Feb. 2000. *www.cs.umb.edu/~dqg.*

[Gu93] Yuri Gurevich. Evolving algebras 1993: Lipari guide. *In* Borger, Ed., *Specification and validation methods,* 1995, pp. 231-243.

[Hol62] John H. Holland. Outline for a logical theory of adaptive systems. *J. ACM* 3 (1962), pp. 297-314.

[Hol87] John H. Holland. Genetic algorithms and classifier systems: foundations and future directions. *In* John J. Grefenstette, Ed., *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Assoc., 1987).

[HH89] John H. Holland, Keith J. Holyoak, Richard E. Nisbett, Paul R. Thagard. *Induction: Processes of Inference, Learning, and Discovery.* MIT Press, 1989.

[JR97] Bart Jacobs, and Jan Rutten. A tutorial on (co)

algebras and (co) induction. *EATCS Bulletin, 62* (1997).

[LT89] Nancy A. Lynch and Mark R. Tuttle. An introduction to input/output automata. *CWI Quarterly* 2(3) (September 1989), pp. 219-246.

[Mil80] Robin Milner. *A calculus of communicating systems.* LNCS 92, Springer, 1980.

[Mil99] Robin Milner. *Communicating and mobile systems*. Cambridge University Press, 1999.

[Min86] Marvin Minsky. *The society of mind.* New York: Simon and Schuster, 1986.

[Pia50] Jean Piaget. *Psychology of intelligence*. Routledge, 1950.

[Res94] Mitchel Resnick. *Turtles, termites, and traffic jams.* MIT Press, 1994.

[RN95] Stuart Russell, Peter Norvig. *Artificial intelligence: a modern approach*. Addison-Wesley, 1995.

[Sim70] Herbert Simon. *The sciences of the artificial.* MIT Press, 1970.

[SL00] Starlogo site at MIT Media Lab. *http://starlogo.www.media.mit.edu/people/starlogo.*

[Tu36] Alan Turing. On computable numbers with an application to the Entscheidungsproblem. *Proc. London Math Society*, 2(42), pp. 173-198, 1936. Reprinted in Martin Davis, Ed., *The Undecidable,* Raven, 1965.

[Tu50] Alan Turing. Computing machinery and intelligence. *Mind, 59* (236) (October, 1950), pp. 433-460.

[VW00] Jan van Leeuwen and Jiri Wiedermann. The Turing machine paradigm in contemporary computing. *In* B. Enquist and W. Schmidt, Eds., *Mathematics Unlimited - 2001 and Beyond*, Springer-Verlag, 2000 (to appear).

[Weg97] Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM 40*:5 (1997).

[Weg99] Peter Wegner. ECOOP'99 Banquet Speech.

[Weg98] Peter Wegner. Interactive foundations of computing. *Theoretical Computer Science 192* (1998).

[WG99a] Peter Wegner and Dina Goldin. Interaction, Computability, and Church's Thesis. Accepted for publication in *Computer Journal*.

[WG99b] Peter Wegner and Dina Goldin. Mathematical models of interactive computing. Brown University Technical Report CS 99-13.

[WG99c] Peter Wegner and Dina Goldin. Coinductive models of finite computing agents. *Electronic notes in Theoretical Computer Science* 19 (March 1999).