AI, Granular Computing, and Automata with Structured Memory

MARK BURGIN and ALLEN KLINGER
Department of Computer Science
University of California, Los Angeles
405 Hilgard Ave., Los Angeles, CA 90095
USA
klinger@cs.ucla.edu

Abstract: - Examining artificial intelligence (AI) problems with the aid of a granular-computing mathematical model. The model is based on Turing machines and recursive algorithms. Through extension to inductive Turing machines, we add super-recursive algorithms enhanced with structured memory to these situations. We consider such AI problems as machine learning, text recognition and advanced computation in the context of granular computational models. Using the elaborated model, we demonstrate that granulation gives a powerful means for learning and computing. The paper shows that it provides both for increasing efficiency of computation and for extending the scope of what is computable, decidable and learnable information.

Key-Words: - artificial intelligence, granular computing, structured memory, machine learning, concurrency

1 Introduction
A granule is a commonly understood characteristic of information. It includes items such as classes, clusters, subsets, components, blocks, groups, and intervals. Granular Computing signifies making use of such elements to construct efficient means to deal with huge amounts of data, information and knowledge. In other words, it describes a computational paradigm for complex applications. (Data usually indicates numeric quantities; information may add to that probabilistic qualities; knowledge indicates difficult-to-quantify understanding that may come from human experience.)

The basic notions and principles of granular computing have appeared in a variety of fields, though under different names. For example: artificial intelligence-AI: granularity; programming: information-hiding; theoretical computer science: divide and conquer algorithms; cluster analysis: interval computing; fuzzy/rough-set theory: neutrosophic computing; quotient space theory: belief functions; machine learning: databases. However, the usual computer science theoretical models: Turing machines, partial recursive functions, random access machines, neural networks, cellular automata and others, do not provide efficient means for granular computing modeling. Here we suggest using Turing machines with structured memory and inductive Turing machines with structured memory as theoretical models of granular computing. We also explain how to adapt these advanced automata to problems of AI. Structured memory provides a means for building multilevel computing architectures. Such architectures according to Bargiela and Pedrycz [4], are inherent in, and essential for, granular computing. Utilizing granular computing models developed in this paper, we study how granulation can improve efficiency and extend power of computation and machine learning. It is demonstrated that granulation gives powerful means for computing, providing for increasing efficiency of computation and learning (Theorem 1), as well as for extending the scope of computable and learnable information (Theorems 2 and 3).

2 Machines with Structured Memory
Automata with structured memory can organize granulation of data, knowledge and tools for their representation utilizing the structure of the automaton memory. Modern neurobiological studies show that automata with structured memory provide relevant models for representation of learning processes in the brain. According to Knudsen [16], learning is a balance of innate and experiential influences. The capacity of a network to learn from experience is limited by innate factors that establish and refine initial patterns of connectivity [17]. These patterns represent specific substructures (granules) in the structured memory of a learning system. In biological systems, such patterns of connectivity can contain remarkable specificity, imparting a high degree of functionality that reflects many generations of selections [19]. This gives evidence to the hypothesis that knowledge and experience are accumulating in structures of memory.

Here we consider such automata with structured memory as Turing machines and inductive Turing
The structure of a Turing machine or inductive Turing machine, as an abstract automaton, is built of three components: \textit{hardware}, \textit{software}, and \textit{infware}. Infware is data and knowledge processed by the machine. Computer infware consists of information, or more exactly, data processed by the computer. An (inductive) Turing machine \( M \) is an abstract automaton, which works with symbolic information in the form of words. Thus, formal languages with which \( M \) works constitute its infware.

The language \( L \) of \( M \) consists of three parts: the \textit{input} language \( L_i \), \textit{working} language \( L_w \), and \textit{output} language \( L_o \). Each has the following structure \( L_X = (A_X, R_X, L_X) \) where \( A_X \) is the alphabet, \( R_X \) is the set of generating rules, and \( L_X \) is the set of all words of the language \( L_X \) where \( X \) is one of the symbols \( I \), \( O \) or \( W \). Usually generating rules for formal languages consist of one operation called concatenation.

As we know, computer hardware consists of all devices (the processor, system of memory, display, keyboard, etc.) that constitute the computer. In a similar way, the \textit{hardware} or \textit{device} \( D \) of the (inductive) Turing machine \( M \) with a structured memory has three components: a \textit{control device} \( A \), which is a finite automaton and controls performance of the machine \( M \); a \textit{processor} or \textit{operating device} \( H \), which corresponds to one or several \textit{heads} of a conventional Turing machine; and the \textit{memory} \( E \), which corresponds to the \textit{tape} or tapes of a conventional Turing machine. For instance, the memory of the simplest inductive Turing machine consists of three linear tapes, and the operating device consists of three heads, each of which is the same as the head of a Turing machine and works with the corresponding tapes [10].

The control device \( A \) has the \textit{state structure} or \textit{configuration} \( S = (q_0, Q, F) \) where \( Q \) is the set of states or the \textit{state space} of \( A \) and of \( M \), \( q_0 \) is an element from \( Q \) that is called the \textit{start} or \textit{initial state}, and \( F \) is a subset of \( Q \) that is called the set of \textit{final} (in some cases, \textit{accepting}) states of \( M \). It is possible to consider a system \( Q_0 \) of start states from \( Q \), but this does not change the computing power of an inductive Turing machine. The automaton \( A \) regulates: the state of the whole machine \( M \), the processing of information by \( H \), and the storage of information in the memory \( E \).

The memory \( E \) is divided into different but, as a rule, uniform cells. It is structured by a system of relations that provide connections or ties between cells. Each cell can contain a symbol from an alphabet of the languages of the machine \( M \) or it can be empty. Formally, \( E = (P, W, K) \) where \( P \) is the set of all cells from \( E \), \( W \) is the set of connection types, and \( K \subset P \times P \) is the binary relation on \( P \) that provides connections between cells. In such a way, \( K \) structures the memory \( E \). Each of the sets \( P \) and \( K \) is also structured. The set \( P \) is enumerated, that is, a one-to-one mapping \( v \) of \( P \) into the set \( N \) of all natural numbers is given. A type is assigned to each connection from \( K \) by the mapping \( \tau: K \rightarrow W \).

In a general case, cells from the set \( P \) also may be of different types. This stratification (or granulation) is represented by the mapping \( \tau: P \rightarrow V \) where \( V \) is the set of cell types. Different types of cells may be used for storing different kinds of symbols. For example, binary cells, which have type \( B \), store bits of information represented by symbols 1 and 0. Byte cells (type \( BT \)) store information represented by strings of eight binary digits. Symbol cells (type \( SB \)) store symbols of the alphabet(s) of the machine \( M \). Cells in conventional Turing machines have this type. Natural number cells, which have type \( NN \), are used in random access machines [1]. Cells in the memory of quantum computers (type \( QB \)) store q-bits or quantum bits. When different kinds of devices are combined into one, this new device has several types of memory cells. In addition, different types of cells facilitate modeling the brain neuron structure by inductive Turing machines [10].

Likewise, the set of cells \( P \) is divided into three disjoint parts \( P_I \), \( P_W \), and \( P_O \), where \( P_I \) is the \textit{input} registers, \( P_W \) is the \textit{working} memory, and \( P_O \) is the \textit{output} registers of \( M \). Correspondingly, \( K \) is divided into three parts \( K_I \), \( K_W \), and \( K_O \) which define connections between the cells from \( P_I \), \( P_W \), and \( P_O \).

It is possible to realize an arbitrary structured memory, using only one linear one-sided tape \( L \). To do this, the cells of \( L \) are enumerated in the natural order from the first one to infinity. Then \( L \) is decomposed into three parts according to the parts \( P_I \), \( P_W \), and \( P_O \) of the structured memory. After this nonlinear connections between cells are installed according to the relation \( K \) and the mapping \( \tau: K \rightarrow W \). When the machine \( M \) with this memory works, the head/processor is not moving to the right or to the left cell from a given cell, but uses the installed nonlinear connections.

Such realization of the structured memory allows us to consider an inductive Turing machine with a structured memory as an inductive Turing machine with conventional tapes in which additional connections are established. This approach has many advantages. One of them is that inductive Turing machines with a structured memory can be treated as multitape automata that have additional structure on their tapes. Then it is conceivable to study...
different ways to construct this structure. In addition, this representation of memory allows us to consider any configuration in the structured memory $E$ as a word written on this unstructured tape.

If we look at other devices of the machine $M$, we can see that the processor $H$ performs information processing in $M$. However, in comparison to computers, this operational device performs very simple operations. When $H$ consists of one unit, it can change a symbol in the cell that is observed by $H$, and go from this cell to another using a connection from $K$. This is exactly what the head of a Turing machine does.

It is possible that the processor $H$ consists of several processing units similar to heads of a multihread Turing machine. This allows in a natural way one to model various real and abstract computing systems by inductive Turing machines. Examples of such systems are: multiprocessor computers; Turing machines with several tapes; networks, grids and clusters of computers; cellular automata; neural networks; and systolic arrays. However, such representation of information processing systems is not always efficient. This is why other models of information processing systems have been constructed, and are and will be utilized.

Connections between the control device $A$ and the processor $H$ may be differently organized: the processor $H$ can have rigid or flexible connections or be autonomous.

We know that programs constitute computer software and tell the system what to do (and what to not do). The software $R$ of the machine $M$ is also a program; it is in the form of simple rules:

$$q_ha_i \rightarrow aq_hc \quad (1)$$

Here $q_h$ and $q_i$ are states of $A$, $a_i$ and $a_h$ are symbols of the alphabet of $M$, and $c$ is a type of connections.

Like Turing machines, inductive Turing machines can be deterministic and nondeterministic. For a deterministic inductive Turing machine, there is at most one connection of any type from any cell. In a nondeterministic inductive Turing machine, several connections of the same type may go from some cells, connecting it with (different) other cells. If there is no connection of this type going from the cell that is observed by $H$, then $H$ stays in the same cell. There may be connections of a cell with itself. Then $H$ also stays in the same cell. It is possible that $H$ observes an empty cell. To represent this situation, we use the symbol $\Lambda$. Thus, it is possible that some elements $a_i$ and/or $a_h$ in the rules from $R$ are equal to $\Lambda$ in the rules of both types. Such rules describe situations when $H$ observes an empty cell and/or when $H$ simply erases the symbol from some cell, writing nothing in it.

When the processor $H$ consists of several processing units or heads, there are several functioning modes:

- Uniform synchronized processing: (processor units function synchronously): At each step of $M$ each unit performs one operation; they all are controlled by the same system of rules.
- Uniform concurrent processing: (processor units function concurrently): Units perform operations independently of one another, but all of them are controlled by the same system of rules.
- Specialized synchronized processing: Each processor unit has its own system of rules, but all of them function synchronously, i.e., at each step of $M$ each unit performs one operation.
- Specialized concurrent processing: Each processor unit has its own system of rules and they perform operations independently of one another.

In what follows, we consider for simplicity only the case when the processor $H$ consists of one unit and $M$ always starts functioning from the same state. Thus, the functioning of the inductive Turing machine $M$ begins when the control device $A$ is in the start state $q_0$, the working and output memories are empty, and the processor $H$ observes such a cell in the input register $P_I$ that this cell contains some symbol and has the least number of all non-empty cells in the input register. It is possible that nothing is written in the input register $P_I$. In this case, $H$ observes an arbitrary cell. When $H$ observes an empty cell, we denote its content by the symbol $\Lambda$.

A general step of the machine $M$ has the following form. At the beginning of any step, the processor $H$ observes some cell with a symbol $a_i$ ($\Lambda$ for an empty cell) and the control device $A$ is in some state $q_a$.

Then the control device $A$ and/or the processor $H$ choose from the system $R$ of rules the rule $r$ with the left part equal to $q_\Lambda a_i$ and perform the operation prescribed by this rule. If there is no rule in $R$ with such left part, the machine $M$ stops functioning. If there are several rules with the same left part, $M$ works as a nondeterministic Turing machine performing all possible operations. When $A$ comes to one of the final states from $F$, the machine $M$ also stops functioning. In all other cases, it continues operation without stopping.

For an abstract automaton, as well as for a computer, two things are important. Specifically, not only how it functions, but also how it obtains its results. In contrast to Turing machines, inductive Turing machines obtain results even in the case when their operation is not terminated. This results...
in essential increase of performance abilities of systems of algorithms.

The computational result of the inductive Turing machine \( M \) is the word that is written in the output register \( P_0 \) of \( M \): when \( M \) halts while its control device \( A \) is in some final state from \( F \), or when \( M \) never stops but at some step of computation the content of the output register \( P_0 \) becomes fixed and does not change although the machine \( M \) continues to function. In all other cases, \( M \) gives no result. Thus, inductive Turing machines produce their results in a finite number of steps without stopping, while Turing machines have to stop to produce the computational result.

Usually memory of information processing systems is stratified and each stratum is a kind of a granule. The complete triadic stratification of process complexity explicates a more developed three-type-gradation of memory: operation memory, short-term memory, and long-term memory.

Operation memory performs information storage for separate steps/operations. Consequently, it stores information for the least time.

Short-term memory performs information storage for separate subprocesses. Consequently, it stores information for longer time.

Long-term memory performs information storage for the whole process. Consequently, it stores information for the longest time.

Usually, in artificial intelligence (cf., for instance, [18]) and psychology (cf., for instance, [2]), only the two latter types are considered. However, a more advanced model of the mind contains all three parts [3, 14, 20]. The model portrays the mind as containing three memory stores: sensory, short-term, and long-term. Each store is characterized by its function (the role it plays in the overall workings of the mind), its capacity (the amount of information it can hold at any given instant), and its duration (the length of time it can hold any item of information).

Computers also have three levels of memory: registers, which constitute a special holding area of the CPU for the numbers the ALU uses for computation; primary storage, which is electronic circuitry that holds data and program instructions until it is their turn to be processed; and additional storage media and devices as a long-term memory [18]. Today two types of long-term storage media are the main ones in use: magnetic and optical. Primary storage includes: RAM, CMOS memory, and ROM. Magnetic storage includes: floppy disks, hard disks, tapes, and Bernoulli disks. Optical storage includes CD-ROM and WORM.

Memory of a Turing machine is its tape(s). In addition, it is possible to store information in states.

Inductive Turing machines have many more possibilities for memory stratification. They are able to realize all three types of computer memory, as well as their subtypes and new types of memory (e.g., quantum memory).

Experience of an individual or society is usually considered as a big data and knowledge base. Such knowledge includes not only declarative knowledge, but also model, procedural, and problem knowledge. Knowledge, according to contemporary understanding is a system in which connections play a central role. Thus, a structured memory of an inductive Turing machine provides efficient means to represent these connections and the knowledge system. At the same time, experience also includes skills and understanding. These attributes cannot be completely reduced to explicit knowledge – they are mainly related to implicit knowledge [9].

Experience is a kind of implicit knowledge. Examples of implicit knowledge are: weights of neurons in neural networks, connections in a structured memory with a dynamic structure, and connections between nodes/automata in virtual or potential grid automata [10].

### 3 Granulation in Structured Memory

To develop a technique for granulation in memory, we need a formal model of granulation. Informally, granulation consists of combining elements from some domain into groups called granules and operating with these granules instead of separate elements. This understanding encompasses many more concrete kinds of granulation: with rough sets, with neighborhood systems, with fuzzy neighborhood systems, etc. [17].

Usually there are different relations between elements from the initial domain, as well as a possibility to perform operations with these elements. This brings us to the concept of an algebraic system.

Granulation as a process is formally represented by a system projection in which system ties between elements induce system ties between granules. Mathematical model of granulation is a partial epimorphism of algebraic systems that is not a one-to-one mapping, i.e., not an isomorphism. An epimorphism is a mapping \( p: A \rightarrow B \) of an algebraic system \( A \) onto an algebraic system \( B \) such that all relations and operations are preserved. In a partial epimorphism not all relations and operations are preserved.

In the context of this mathematical model of granulation, a granule is an element of \( B \). According
to the triarchic theory of granular computing [21], properties of a granule belong to one of the three basic groups: internal properties, external properties, and contextual properties. A granule \( b \) is treated both as a cluster of individual elements characterized by internal properties and as an inseparable whole characterized by its external properties. The existence of a granule is only meaningful if elements in a granule are themselves granules, and a granule can also be an element of another granule. Three groups of properties are mathematically represented in the following way:

1. Internal properties are system properties formed by the inverse image \( p^{-1}(b) \) of granule \( b \) from \( B \).
2. External properties are properties of the element \( b \) with respect to operations that are performed only with \( b \) and relations that involve only \( b \) but not other elements from \( B \).
3. Contextual properties are properties of \( b \) with respect to all other elements from \( B \).

Granulation in a structured memory goes in the following way. Each granule \( g \) is associated with its memory space \( S(g) \). This correspondence determines memory granulation \( m \) both in informal [17, 21] and formalized here sense. In the mathematical interpretation, a structured memory is an algebraic system and its stratification (granulation) determines a projection (partial epimorphism) onto the quotient algebraic system.

All data and knowledge that constitute the granule \( g \) are allocated to the space \( S(g) \). The name "\( g \)" of the granule \( g \) is also represented in the memory and all interactions with the data and knowledge from the granule \( g \) go through its name "\( g \)". This name becomes a connecting link with and a port for the inner content of the granule.

In such a way, the memory is stratified (granulated) and this stratification (granulation) is represented by the following diagram in the category of algebraic systems.

\[
\begin{array}{ccc}
D & \xrightarrow{\mu} & E \\
\downarrow{p} & & \downarrow{m} \\
G & \xrightarrow{v} & QE \\
\end{array}
\]

Here \( E \) is a structured memory, \( QE \) is its stratification (the quotient algebraic system), \( D \) is the initial data and/or knowledge system, and \( G \) is the granulated data and/or knowledge system.

### 3 Knowledge Granulation and Stratified Learning

Learning consists of diverse forms of activity. Further one can consider cognition as a kind of learning. It is learning from the environment or nature. Such traditional forms as testing, verification, and detection, as well as such new forms as data mining, web search, and client monitoring are also kinds of learning. Granulation can help and optimize learning procedures and algorithms.

Results from [6, 7] show that granulation can essentially improve efficiency. Based on these results and using reflexive Turing machines and reflexive inductive Turing machines, we can prove the following theorem.

**Theorem 1.** Granulation of information (data) and rules of computation based on a structured memory allows a reflexive Turing machine or a reflexive inductive Turing machine \( M \) to make learning \( n^k \) times more efficient, i.e., to decrease in \( n^k \) times the time of learning of a given algorithm.

To understand abilities of granulation in computational and learning processes, it is necessary to discern different modes of computation or learning. We consider here three basic modes [10].

1. **Recursive mode**, when the machine gives the final result after a finite number of steps and after this stops or informs when the result is obtained.
2. **Inductive mode**, when the machine gives the final result after a finite number of steps but it does not always after this stop the process of computation or informs when the result is obtained.
3. **Limit mode**, when the machine gives the partial result after a finite number of steps and the final result is the limit of these partial results.

Usually recursive mode is restricted even more, becoming polynomially bounded in time, i.e., when time of computation is bounded by some polynomial, or in space, i.e., when the used memory is bounded by some polynomial.

Results from [11] show that granulation does not improve learning potential, i.e., what is possible to learn in general without extra restrictions, when the learning system works in the recursive or polynomially bounded mode. However, when the learning system is functioning in the inductive mode, granulation allows the system to expand immensely the scope of learning. It is known [13] that an algorithm working in the recursive mode can learn only finite sequences of symbols or functions on finite sets. For the inductive mode, we have much stronger results.
**Theorem 2.** In the inductive mode, a learning system (algorithm) can learn or recognize any recursively computable (enumerable) sequences of symbols and using granulation, any recursively computable function.

Granulation becomes very powerful when results of one machine (automaton) become granules for computation of another machine (automaton). This is a new type of granulation, which we call experience granulation. This approach assumes that learning goes in stages and knowledge obtained at the previous stage is granulated and used for learning at the next stage. Granulation of knowledge is based on its properties obtained in [9].

To measure abilities of automata with and without granulation, we use the arithmetical hierarchy of sets [10]. This hierarchy consists of the countable number of levels. Recursively decidable sets form the zero level of this hierarchy. Recursive computations allow one to build the next level of the hierarchy. As it was proved by Gold [12], inductive computations allow automata to reach only two next levels.

Based on results from [6], we can prove the following theorem.

**Theorem 3.** Inductive Turing machines with structured memory and experience granulation in this memory can span the whole arithmetical hierarchy.

### 4 Conclusion

Thus, we have demonstrated how power grows when a learner uses knowledge and experience granulation. It is demonstrated that neither recursive nor subrecursive algorithms give any increase of potentially learnable knowledge with or without granulation. Only super-recursive algorithms, such as inductive Turing machines, allow achieving better results in stratified learning based on experience granulation. At the same time, it is also demonstrated that it is possible to increase efficiency of learning and computation both for recursive and super-recursive automata when an appropriate granulation is performed in a structured memory.

---

**References:**


