

On Contemporary Denotational Mathematics for Computational Intelligence

Yingxu Wang

Theoretical and Empirical Software Engineering Research Centre (TESERC)
International Center for Cognitive Informatics (ICfCI)
Dept. of Electrical and Computer Engineering
Schulich Schools of Engineering, University of Calgary
2500 University Drive, NW, Calgary, Alberta, Canada T2N 1N4
Tel.: (403) 220 6141, Fax: (403) 282 6855
yingxu@ucalgary.ca

Abstract. Denotational mathematics is a category of expressive mathematical structures that deals with high-level mathematical entities beyond numbers and sets, such as abstract objects, complex relations, behavioral information, concepts, knowledge, processes, intelligence, and systems. New forms of mathematics are sought, collectively known as denotational mathematics, in order to deal with complex mathematical entities emerged in cognitive informatics, computational intelligence, software engineering, and knowledge engineering. The domain and architecture of denotational mathematics are presented in this paper. Three paradigms of denotational mathematics, known as concept algebra, system algebra, and Real-Time Process Algebra (RTPA), are introduced. Applications of denotational mathematics in cognitive informatics and computational intelligence are elaborated. A set of case studies is presented on the modeling of iterative and recursive systems architectures and behaviors by RTPA, the modeling of autonomic machine learning by concept algebra, and the modeling of granular computing by system algebra.

Keywords: Denotational mathematics, concept algebra, system algebra, process algebra, RTPA, cognitive informatics, computational intelligence, software engineering, knowledge engineering, embedded relations, incremental relations, big-R notation.

1 Introduction

The history of sciences and engineering shows that many branches of mathematics have been created in order to meet their *abstract*, *rigorous*, and *expressive* needs. These phenomena may be conceived as that new problems require new forms of mathematics [5], [26]. It also indicates that the maturity of a new discipline is characterized by the maturity of its theories denoted in rigorous and efficient mathematical means [42], [43]. Therefore, the entire computing theory, as Lewis and Papadimitriou perceived, is about mathematical models of computers and algorithms [19]. Hence, the entire theory of cognitive informatics, computational intelligence, and software

science is about new mathematical structures for natural and machine intelligence and efficient mathematical means.

Applied mathematics can be classified into two categories known as *analytic* and *denotational* mathematics [37], [42], [43], [46]. The former are mathematical structures that deal with functions of variables and their operations and behaviors; while the latter are mathematical structures that formalize rigorous expressions and inferences of system architectures and behaviors with abstract concepts and dynamic processes. It is observed that all existing mathematics, continuous or discrete, are mainly analytic, seeking unknown variables from known factors according to certain functions. Modern sciences have been mainly using analytic methodologies and mathematics in theory development and problem solving. However, in cognitive informatics and computational intelligence, the need is to formally describe and manipulate software and instructional behaviors in terms of operational logic, timing, and memory manipulation. Therefore, denotational mathematics are sought [37], [42], [43], [46], [48-52], which are able to describe software and intelligent architectures and behaviors rigorously, precisely, and expressively.

Definition 1. *Denotational mathematics* is a category of expressive mathematical structures that deals with high-level mathematical entities beyond numbers and sets, such as abstract objects, complex relations, behavioral information, concepts, knowledge, processes, intelligence, and systems.

The utility of denotational mathematics serves as the means and rules to rigorously and explicitly express design notions and conceptual models of abstract architectures and interactive behaviors of complex systems at the highest level of abstraction, in order to deal with the problems of cognitive informatics and computational intelligence characterized by large scales, complex architectures, and long chains of computing behaviors. Therefore, denotational mathematics is a system level mathematics, in which detailed individual computing behaviors may still be manipulated by conventional analytical mathematics. Typical forms of denotational mathematics [43], [46] are concept algebra [49], system algebra [50], and Real-Time Process Algebra (RTPA) [37], [41], [43], [51], [52].

It is observed in formal linguistics that human and system behaviors can be classified into three categories: to *be*, to *have*, and to *do* [6], [30], [41], [43]. All mathematical means and forms, in general, are an abstract description of these three categories of human and system behaviors and common rules of them. Taking this view, as shown in Table 1, mathematical logic may be perceived as the abstract means for describing “to *be*,” set theory for describing “to *have*,” and functions for describing “to *do*” in classic mathematics.

Table 1 summarized the usages of classic and denotational mathematics, which presents a fundamental view toward the modeling and expression of natural and machine intelligence in general, and software system in particular. Table 1 also indicates that only the logic- and set-based approaches are inadequate to deal with the entire problems in complex software and intelligent systems.

Table 1. Basic Expressive Power and Mathematical Means in System Modeling

Basic expressive power in system modeling	Mathematical means		Usage
	Classic mathematics	Denotational mathematics	
To <i>be</i>	Logic	Concept algebra	Identify <i>objects</i> and <i>attributes</i>
To <i>have</i>	Set theory	System algebra	Describe <i>relations</i> and <i>possession</i>
To <i>do</i>	Functions	RTPA	Describe <i>status</i> and <i>behaviors</i>

This paper presents the contemporary denotational mathematical structures for cognitive informatics and computational intelligence beyond classic mathematical entities, such as information, concepts, knowledge, processes, behaviors, intelligence, systems, distributed objects, and complex relations. The emergence and domain of denotational mathematics are described in Section 2. The paradigms of denotational mathematics, such as concept algebra, system algebra, and RTPA, are introduced in Section 3. Applications of denotational mathematics are demonstrated in Section 4, which covers the modeling of iterative and recursive systems architectures and behaviors by RTPA, the modeling of autonomic machine learning by concept algebra, and the modeling of granular computing by system algebra.

2 The Emergence and Development of Denotational Mathematics

The emergence of denotational mathematics is driven by the practical needs in cognitive informatics, computational intelligence, computing science, software science, and knowledge engineering, because all these modern disciplines study complex human and machine behaviors and their rigorous treatments. This section analyzes the fundamental elements in modeling computing systems and explains why these complex mathematical entities cannot be modeled by simple numbers and sets. This leads to the requirements for denotational mathematics that extends both entities and their manipulations in conventional mathematics. Then, the domain and architecture of denotational mathematics are summarized.

2.1 Fundamental Elements in Modeling Cognitive and Intelligent Systems

It is recognized that the behavioral space of any system or human action is three-dimensional, which encompasses the dimensions of *action*, *time*, and *space* [43]. Correspondingly, there are three fundamental categories of computational behaviors in a software system: a) Computational operations for variable manipulations, b) Timing operations for event manipulation, and c) Space operations for memory manipulation. Therefore, the behavior of a software or intelligent system can, in general, be viewed as a set of behavioral processes with computational operations on time and memory.

Definition 2. A *behavior* of a software or intelligent system, B , is a tuple of its computing operations OPs and observable outcomes and effects that affect or change the states of a system in the environment modeled by all variables and input/output events, as well as related memory structures M over time T , i.e.:

$$\begin{aligned}
 B &\triangleq (OP, T, M) \\
 &= OP \times T \times M
 \end{aligned}
 \tag{1}$$

Behaviors of generic computing systems can be classified as static and dynamic ones as shown in Table 2. In Table 2, a *static behavior* of computing is a process that can be determined at design or compile time; while a *dynamic behavior* of computing is a process specified by given timing requirements that may only be determined at run-time.

Table 2. Characteristics of Computing System Behaviors

No.	Behaviors	Static	Dynamic	Behavioral category
1	System architectures	✓	✓	To be / to have
2	Data objects	✓	✓	To be / to have
3	Dynamic memory allocation		✓	To do
4	Timing		✓	To do
5	Input/output manipulations		✓	To do
6	Event handling		✓	To do
7	Mathematical operations	✓	✓	To do

It is noteworthy in Table 2 that most system behaviors are dynamic or both dynamic and static. Set theories and mathematical logic are found capable to deal with the ‘to be’ and ‘to have’ type static behaviors. However, the dynamic ‘to do’ behaviors in computing have to be manipulated by process algebras, e.g., RTPA. Even for the first two categories of behavioral problems in software and intelligent systems, concept algebra and system algebra are capable to deal with the problems more efficiently than logic and set theories, because they work at a higher level of mathematical entities known as abstract concepts and systems rather than numbers and sets.

2.2 New Problems Require New Forms of Mathematics

The history of sciences and engineering shows that new problems require new forms of mathematics. Software science and computational intelligence are emerging trans-disciplinary enquiries that encompass a wide range of contemporary mathematical entities, which cannot be adequately described by conventional analytic mathematics. Therefore, new forms of mathematics are sought, collectively known as denotational mathematics.

The discussions in Section 2.1 indicate that classic mathematical forms such as sets, logic, and functions are inadequate to deal with the complex and dynamic behavioral problems of software and intelligent systems. The weaknesses of classic mathematics

are in both of their expressive power and manipulation efficiency in the three categories of system descriptivity. The profound problems can be analogized to the evolutions of computing architectures, where, although Turing machines [19] are the most fundamental models for any computing need, von Neumann machines [36] and cognitive machines [38], [39], [44] are required to solve the problems of complex data processing and knowledge processing more efficiently and expressively.

A great extent of effort has been put on extending the capacity of sets and mathematical logic in dealing with the above problems in cognitive informatics and computational intelligence. The former are represented by the proposals of fuzzy sets [60, 62] and rough sets [27]. The latter are represented by the development of temporal logic [29] and fuzzy logic [60]. New mathematical structures are created such as *embedded relations*, *incremental relations*, and the *big-R notation* [37], [48]. More systematically, a set of new denotational mathematical forms [42], [43], [46] are developed known as concept algebra [49], system algebra [50], and RTPA [37], [40], [41], [43], [51], [52]. These new mathematical structures are introduced below, while the three paradigms of denotational mathematics will be elaborated in Section 3.

2.2.1 The Big-R Notation

The *big-R notation* is introduced to deal with the fundamental requirement in computing and software engineering [48], which is proposed first in RTPA [37]. In order to develop a general mathematical model for unifying the syntaxes and semantics of iterations and recursions, their inductive nature may be analyzed as follows.

Definition 3. An *iteration* of a process P can be defined as a series of $n+1$ repetitions, R_i , $1 \leq i \leq n+1$, of P by mathematical induction, i.e.:

$$\begin{aligned} R_0 &= \otimes, \\ R_1 &= P \rightarrow R_0, \\ &\dots \\ R_{n+1} &= P \rightarrow R_n, \quad n \geq 0 \end{aligned} \tag{2}$$

where \otimes denotes a skip, or doing nothing but exit.

Based on Definition 3, the big-R notation can be introduced below.

Definition 4. The *big-R notation*, \mathbf{R} , is a generic mathematical calculus in computing that is used to denote: (a) a finite set of *repetitive* behaviors, or (b) a finite set of recurring architectural constructs, in the following forms, respectively:

$$(a) \quad \mathbf{R}_{\text{exp}\mathbf{BL}=\mathbf{T}}^{\mathbf{F}} P \tag{3}$$

$$(b) \quad \mathbf{R}_{i\mathbf{N}=1}^n P(i\mathbf{N}) \tag{4}$$

where \mathbf{BL} and \mathbf{N} are the type suffixes of Boolean and natural numbers, respectively; \mathbf{T} and \mathbf{F} are the Boolean constants true and false, respectively.

The big-R notation is a new denotational mathematical structure that enables efficient representation and manipulation of iterative and recursive behaviors in system modeling and description. Further description of the type system and a summary of all type suffixes of RTPA will be presented in Section 3.3.

2.2.2 The Embedded Relations

Definition 5. An *embedded relation* r_{ij} is a sequence of left-associated cumulative relations among a series of computing behaviors p_i and p_j , $1 \leq i < n$, $1 < j \leq m = n+1$, i.e.:

$$(\dots((p_1) r_{12} p_2) r_{23} p_3) \dots r_{n-1,n} p_n) = \mathbf{R}_{i=1}^{n-1}(p_i r_{ij} p_j), j=i+1 \quad (5)$$

where $r_{ij} \in \mathfrak{R}$, which is a set of relational process operators of RTPA that will be formally defined in Lemma 6.

The embedded relational operation is a new denotational mathematical structure, which provides a generic mathematic model for any program and software system in computing and intelligent system modeling.

2.2.3 The Incremental Relations

Definition 6. An *incremental union* of two sets of relations R_1 and R_2 , denoted by \boxplus , are a union of R_1 and R_2 plus a newly generated incremental set of relations ΔR_{12} , i.e.:

$$R_1 \boxplus R_2 \triangleq R_1 \cup R_2 \cup \Delta R_{12} \quad (6)$$

where $\Delta R_{12} \not\subseteq R_1 \wedge \Delta R_{12} \not\subseteq R_2$ and $\Delta R_{12} = 2(\#C_1 \bullet \#C_2) \subseteq R_1 \boxplus R_2$.

The incremental relational operation is a new denotational mathematical structure, which provides a generic mathematical model for revealing the fusion principle and system gains during system unions and compositions.

2.3 The Domain and Architecture of Denotational Mathematics

The emergence of new mathematical entities in computing, cognitive informatics, and computational intelligence, as well as the requirements for new mathematical calculi, are the driving forces for seeking new mathematical structures and forms known collectively as denotational mathematics. The domain and architecture of denotational mathematics are illustrated in Fig. 1.

Denotational mathematics is usually in the form of abstract algebra that is a form of mathematics in which a system of abstract notations is adopted to denote relations of abstract mathematical entities and their algebraic operations based on given axioms and laws. Denotational mathematics may be used to denote complex behaviors of humans and intelligent systems, as well as long sequences of inference processes. A wide range of applications of denotational mathematics has been identified, such as cognitive informatics, computational intelligence, software engineering, knowledge engineering, information engineering, autonomic computing, autonomous machine learning, and neural informatics.

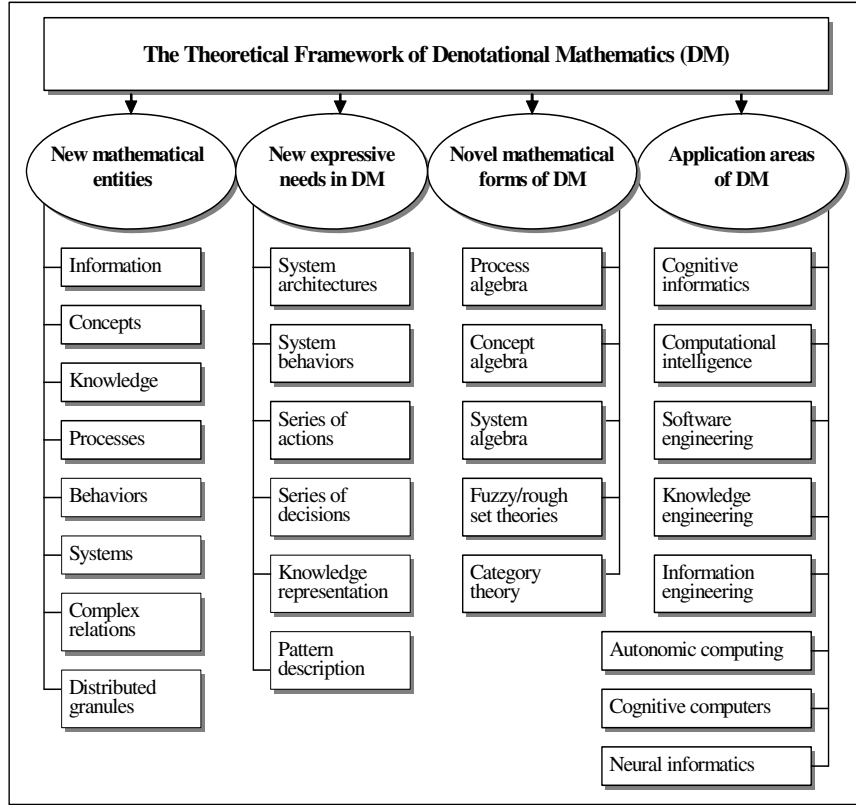


Fig. 1. Architecture of Denotational Mathematics

The following sections will introduce three paradigms of contemporary denotational mathematics. Their applications in cognitive informatics and computational intelligence will be demonstrated, which show how denotational mathematics may greatly improve the expressive power and efficiency in complex system modeling and manipulations.

3 Paradigms of Denotational Mathematics

Algebra is a branch of mathematics in which a system of symbolic abstractions and algebraic operations are adopted to denote variables, relations and their manipulation rules. Extensions of conventional algebra onto more complicated mathematical entities beyond numbers lead to the contemporary denotational mathematics. Three new denotational mathematical forms are created in exploring the mathematical foundations of cognitive informatics and computational intelligence [42], [43], [46]. Within the new forms of descriptive mathematics, *concept algebra* is designed to deal with the new abstract mathematical structure of concepts and their representation and manipulation. RTPA is developed to deal with series of behavioral processes and architectures of

software and intelligent systems. *System algebra* is created for the rigorous treatment of abstract systems and their algebraic operations.

3.1 Concept Algebra

In cognitive informatics, logic, linguistics, psychology, software engineering, knowledge engineering, and computational intelligence, concepts are identified as the basic unit of both knowledge and reasoning [2], [8], [11], [12], [17], [22], [25], [41], [45], [49]. The rigorous modeling and formal treatment of concepts are at the center of theories for knowledge presentation and manipulation [7], [25], [34], [55], [59]. A *concept* in linguistics is a noun or noun-phrase that serves as the subject or object of a *to-be* statement [17], [37], [43]. Concepts in denotational mathematics [37], [49] are an abstract structure that carries certain meaning in almost all cognitive processes such as thinking, learning, and reasoning.

Definition 7. A *concept* is a cognitive unit to identify and/or model a real-world concrete entity and a perceived-world abstract object.

This section describes the formal treatment of abstract concepts and a new mathematical structure known as concept algebra in cognitive informatics and computational intelligence. Before an abstract concept is defined, the semantic environment or context [11], [12], [17], [23], [59] in a given language, is introduced.

Definition 8. Let \mathcal{O} denote a finite nonempty set of *objects*, and \mathcal{A} be a finite nonempty set of *attributes*, then a *semantic environment* or *context* $\Theta_{\mathcal{C}}$ is denoted as a triple, i.e.:

$$\begin{aligned} \Theta_{\mathcal{C}} &\triangleq (\mathcal{O}, \mathcal{A}, \mathcal{R}) \\ &= \mathcal{R}: \mathcal{O} \rightarrow \mathcal{O} \mid \mathcal{O} \rightarrow \mathcal{A} \mid \mathcal{A} \rightarrow \mathcal{O} \mid \mathcal{A} \rightarrow \mathcal{A} \end{aligned} \quad (7)$$

where \mathcal{R} is a set of relations between \mathcal{O} and \mathcal{A} , and \mid demotes alternative relations.

Definition 9. An *abstract concept* c on $\Theta_{\mathcal{C}}$ is a 5-tuple, i.e.:

$$c \triangleq (\mathcal{O}, \mathcal{A}, R^c, R^i, R^o) \quad (8)$$

where

- \mathcal{O} is a finite nonempty set of objects of the concept, $\mathcal{O} = \{o_1, o_2, \dots, o_m\} \subseteq \mathbb{P}\mathcal{O}$, where $\mathbb{P}\mathcal{O}$ denotes a power set of \mathcal{O} .
- \mathcal{A} is a finite nonempty set of attributes, $\mathcal{A} = \{a_1, a_2, \dots, a_n\} \subseteq \mathbb{P}\mathcal{A}$.
- $R^c = \mathcal{O} \times \mathcal{A}$ is a set of internal relations.
- $R^i \subseteq \mathcal{A}' \times \mathcal{A}$, $\mathcal{A}' \sqsubseteq \mathcal{C}' \wedge \mathcal{A} \sqsubseteq c$, is a set of input relations, where \mathcal{C}' is a set of external concepts, $\mathcal{C}' \subseteq \Theta_{\mathcal{C}}$. For convenience, $R^i = \mathcal{A}' \times \mathcal{A}$ may be simply denoted as $R^i = \mathcal{C}' \times c$.
- $R^o \subseteq c \times \mathcal{C}'$ is a set of output relations.

Concept algebra is an abstract mathematical structure for the formal treatment of concepts and their algebraic relations, operations, and associative rules for composing complex concepts.

Definition 10. A *concept algebra* CA on a given semantic environment Θ_C is a triple, i.e.:

$$CA \triangleq (C, OP, \Theta_C) = (\{O, A, R^c, R^i, R^o\}, \{\bullet_r, \bullet_c\}, \Theta_C) \quad (9)$$

where $OP = \{\bullet_r, \bullet_c\}$ are the sets of *relational* and *compositional* operations on abstract concepts.

Lemma 1. The *relational operations* \bullet_r in concept algebra encompass 8 comparative operators for manipulating the algebraic relations between concepts, i.e.:

$$\bullet_r \triangleq \{\leftrightarrow, \Leftrightarrow, \prec, \succ, =, \cong, \sim, \triangleq\} \quad (10)$$

where the relational operators stand for *related*, *independent*, *subconcept*, *superconcept*, *equivalent*, *consistent*, *comparison*, and *definition*, respectively.

Lemma 2. The *compositional operations* \bullet_c in concept algebra encompass 9 associative operators for manipulating the algebraic compositions among concepts, i.e.:

$$\bullet_c \triangleq \{\overset{-}{\Rightarrow}, \overset{+}{\Rightarrow}, \overset{\sim}{\Rightarrow}, \Rightarrow, \uplus, \updownarrow, \Leftarrow, \vdash, \mapsto\} \quad (11)$$

where the compositional operators stand for *inheritance*, *tailoring*, *extension*, *substitute*, *composition*, *decomposition*, *aggregation*, *specification*, and *instantiation*, respectively.

Concept algebra provides a denotational mathematical means for algebraic manipulations of abstract concepts. Concept algebra can be used to model, specify, and manipulate generic “to be” type problems, particularly system architectures, knowledge bases, and detail-level system designs, in cognitive informatics, computational intelligence, computing science, software engineering, and knowledge engineering. Detailed relational and compositional operations of concept algebra may be referred to [42], [49].

3.2 System Algebra

Systems are the most complicated entities and phenomena in abstract, physical, information, and social worlds across all science and engineering disciplines. The system concept can be traced back to the 17th Century when R. Descartes (1596-1650) noticed the interrelationships among scientific disciplines as a system. Then, the general system notion was proposed by Ludwig von Bertalanffy in the 1920s [35], followed by the theories of system science [3], [4], [9], [13], [18], [31]. Further, there are proposals of complex systems theories [18], [61], fuzzy theories [60], [61], and chaos theories [10], [32]. Yingxu Wang found that, because of their extremely wide and frequent usability, systems may be treated rigorously as a new mathematical structure beyond conventional mathematical entities known as the *abstract systems* [50]. Based on this view, the concept of abstract systems and their mathematical models are introduced below.

Definition 11. An *abstract system* is a collection of coherent and interactive entities that has stable functions and a clear boundary with the external environment.

An abstract system forms the generic model of various real-world systems and represents the most common characteristics and properties of them. For instance, the granularity of granular computing can be explained by the following lemma in the abstract system theory.

Lemma 3. The *generality principle of system abstraction* states that a system can be represented as a whole in a given level k of reasoning, $1 \leq k \leq n$, without knowing the details at levels below k .

Definition 12. Let \mathcal{C} be a finite nonempty set of *components*, and \mathcal{B} a finite nonempty set of *behaviors*, then the *universal system environment* \mathfrak{U} is denoted as a triple, i.e.:

$$\begin{aligned} \mathfrak{U} &\triangleq (\mathcal{C}, \mathcal{B}, \mathcal{R}) \\ &= \mathcal{R}: \mathcal{C} \rightarrow \mathcal{C} \mid \mathcal{C} \rightarrow \mathcal{B} \mid \mathcal{B} \rightarrow \mathcal{C} \mid \mathcal{B} \rightarrow \mathcal{B} \end{aligned} \quad (12)$$

where \mathcal{R} is a set of relations between \mathcal{C} and \mathcal{B} , and \mid demotes alternative relations.

Abstract systems can be classified into two categories known as the *closed* and *open* systems. Most practical and useful systems in nature are open systems in which there are interactions between the system and its environment. That is, they need to interact with external world known as the *environment* Θ , $\Theta \sqsubseteq \mathfrak{U}$, in order to exchange energy, matter, and/or information. Such systems are called open systems. Typical interactions between an open system and the environment are inputs and outputs.

Definition 13. An *open system* S on \mathfrak{U} is a 7-tuple, i.e.:

$$S \triangleq (\mathcal{C}, R^c, R^i, R^o, B, \Omega, \Theta) \quad (13)$$

where

- \mathcal{C} is a finite nonempty set of *components* of the system, $\mathcal{C} = \{c_1, c_2, \dots, c_n\} \subseteq \mathfrak{PC} \sqsubseteq \mathfrak{U}$.
- R is a finite nonempty set of *relations* between pairs of the components in the system, $R = \{r_1, r_2, \dots, r_m\} \subseteq \mathcal{C} \times \mathcal{C}$.
- $R^c = \mathcal{C} \times \mathcal{C}$ is a set of *internal relations*.
- $R^i \subseteq \mathcal{C}_\Theta \times \mathcal{C}$ is a set of external *input relations*, $\mathcal{C}_\Theta \subseteq \mathfrak{PC} \sqsubseteq \mathfrak{U}$.
- $R^o \subseteq \mathcal{C} \times \mathcal{C}_\Theta$ is a set of external *output relations*.
- B is a set of *behaviors* (or functions), $B = \{b_1, b_2, \dots, b_p\} \subseteq \mathfrak{PB} \sqsubseteq \mathfrak{U}$.
- Ω is a set of *constraints* on the memberships of components, the conditions of relations, and the scopes of behaviors, $\Omega = \{\omega_1, \omega_2, \dots, \omega_q\}$.
- Θ is the *environment* of S with a nonempty set of components \mathcal{C}_Θ outside \mathcal{C} , i.e., $\Theta = \mathcal{C}_\Theta \sqsubseteq \mathfrak{PC} \sqsubseteq \mathfrak{U}$.

System algebra is an abstract mathematical structure for the formal treatment of abstract and general systems as well as their algebraic relations, operations, and associative rules for composing and manipulating complex systems [43], [50].

Definition 14. A *system algebra* SA on a given universal system environment \mathcal{U} is a triple, i.e.:

$$SA \triangleq (S, OP, \Theta) = (\{C, R^c, R^l, R^o, B, \Omega\}, \{\bullet_r, \bullet_c\}, \Theta) \quad (14)$$

where $OP = \{\bullet_r, \bullet_c\}$ are the sets of *relational* and *compositional* operations, respectively, on abstract systems as defined below.

Definition 15. The *relational operations* \bullet_r in system algebra encompass 6 comparative operators for manipulating the algebraic relations between abstract systems, i.e.:

$$\bullet_r \triangleq \{\Leftrightarrow, \leftrightarrow, \sqcap, =, \sqsubseteq, \supseteq\} \quad (15)$$

where the relational operators stand for *independent*, *related*, *overlapped*, *equivalent*, *subsystem*, and *supersystem*, respectively.

Definition 16. The *compositional operations* \bullet_c in system algebra encompass 9 associative operators for manipulating the algebraic compositions among abstract systems, i.e.:

$$\bullet_c \triangleq \{\Rightarrow, \overset{-}{\Rightarrow}, \overset{+}{\Rightarrow}, \overset{\sim}{\Rightarrow}, \boxminus, \boxplus, \uplus, \Leftarrow, \vdash\} \quad (16)$$

where the compositional operators stand for system *inheritance*, *tailoring*, *extension*, *substitute*, *difference*, *composition*, *decomposition*, *aggregation*, and *specification*, respectively.

System algebra provides a denotational mathematical means for algebraic manipulations of all forms of abstract systems. System algebra can be used to model, specify, and manipulate generic “to be” and “to have” type problems, particularly system architectures and high-level system designs, in cognitive informatics, computational intelligence, computing science, software engineering, and system engineering. It will be demonstrated in Section 4.3 that the abstract system and system algebra are an ideal model for rigorously describing both the structures and behaviors of granules in granular computing. Detailed relational and compositional operations on abstract systems may be referred to [50].

3.3 Real-Time Process Algebra (RTPA)

A key metaphor in system modeling, specification, and description is that a software and intelligent system can be perceived and described as the *composition* of a set of interacting *processes*. Hoare [15], [16], Milner [24], and others developed various algebraic approaches to represent communicating and concurrent systems, known as process algebra. A *process algebra* is a set of formal notations and rules for describing algebraic relations of software engineering processes. RTPA [37], [40], [43], [51], [52] is a real-time process algebra that can be used to formally and precisely describe and specify architectures and behaviors of human and software systems.

Definition 17. A process P is an embedded relational composition of a list of n meta-statements p_i and p_j , $1 \leq i < n$, $1 < j \leq m = n+1$, according to certain composing relations r_{ij} , i.e.:

$$P = \mathbf{R}_{i=1}^{n-1}(p_i r_{ij} p_j), j = i+1 \quad (17)$$

$$= (\dots(((p_1) r_{12} p_2) r_{23} p_3) \dots r_{n-1,n} p_n)$$

where the big-R notation [48] is adopted that describes the nature of processes as the building blocks of programs.

Definition 17. indicates that the mathematical model of a process is a cumulative relational structure among basic computing operations, where the simplest process is a single computational statement.

Definition 18. RTPA is a denotational mathematical structure for algebraically denoting and manipulating system behavioural processes and their attributes by a triple, i.e.:

$$RTPA \triangleq (\mathfrak{T}, \mathfrak{P}, \mathfrak{R}) \quad (18)$$

where \mathfrak{T} is a set of 17 primitive types for modeling system architectures and data objects, \mathfrak{P} a set of 17 meta-processes for modeling fundamental system behaviors, and \mathfrak{R} a set of 17 relational process operations for constructing complex system behaviors.

Lemma 4. The *primary types of computational objects* state that the RTPA type system \mathfrak{T} encompasses 17 primitive types elicited from fundamental computing needs, i.e.:

$$\mathfrak{T} \triangleq \{\mathbf{N}, \mathbf{Z}, \mathbf{R}, \mathbf{S}, \mathbf{BL}, \mathbf{B}, \mathbf{H}, \mathbf{P}, \mathbf{TI}, \mathbf{D}, \mathbf{DT}, \mathbf{RT}, \mathbf{ST}, @e\mathbf{S}, @t\mathbf{TM}, @int\odot, @s\mathbf{BL}\} \quad (19)$$

where the primary types of RTPA denote *natural number, integer, real number, string, Boolean variable, byte, hyper-decimal, pointer, time, date, date-time, run-time type, system type, event, timing-event, interrupt-event*, and *system status*, respectively.

Definition 19. A *meta-process* in RTPA is a primitive computational operation that cannot be broken down to further individual actions or behaviors.

A meta-process is an elementary process that serves as a basic building block for modeling software behaviors. *Complex processes* can be composed from meta-processes using *process relational operations*. In RTPA, a set of 17 meta-processes has been elicited from essential and primary computational operations commonly identified in existing formal methods and modern programming languages [1], [14], [21], [56], [57].

Lemma 5. The RTPA meta-process system \mathfrak{P} encompasses 17 fundamental computational operations elicited from the most basic computing needs, i.e.:

$$\mathfrak{P} \triangleq \{ :=, \blacklozenge, \Rightarrow, \Leftarrow, \Leftarrow, \succ, \prec, \triangleright, \triangleleft, @, \triangle, \Uparrow, \Downarrow, !, \otimes, \boxtimes, \S \} \quad (20)$$

where the meta-processes of RTPA stand for *assignment, evaluation, addressing, memory allocation, memory release, read. write, input, output, timing, duration, increase, decrease, exception detection, skip, stop, and system*, respectively.

Definition 20. A *process relation* in RTPA is an algebraic operation and a compositional rule between two or more meta-processes in order to construct a complex process.

A set of 17 fundamental process relations has been elicited from fundamental algebraic and relational operations in computing in order to build and compose complex processes in the context of real-time software systems.

Lemma 6. The software composing rules state that the *RTPA process relation system* \mathfrak{R} encompasses 17 fundamental algebraic and relational operations elicited from basic computing needs, i.e.:

$$\mathfrak{R} \triangleq \{ \rightarrow, \curvearrowright, |, | \dots | \dots, \mathbf{R}^*, \mathbf{R}^+, \mathbf{R}^i, \circ, \multimap, \parallel, \text{\textcircled{R}}, \text{\textcircled{L}}, \gg, \leftarrow, \hookrightarrow, \hookrightarrow_e, \hookrightarrow_i \} \quad (21)$$

where the relational operators of RTPA stand for *sequence, jump, branch, while-loop, repeat-loop, for-loop, recursion, function call, parallel, concurrence, interleave, pipeline, interrupt, time-driven dispatch, event-driven dispatch, and interrupt-driven dispatch*, respectively.

The generic program model can be established by a formal treatment of statements, processes, and complex processes from the bottom-up in the program hierarchy.

Definition 21. A *program* \wp is a composition of a finite nonempty set of m processes according to the time-, event-, and interrupt-based process dispatching rules, i.e.:

$$\wp = \mathbf{R}_{k=1}^m (@ e_k \hookrightarrow P_k) \quad (22)$$

Definitions 17 and 21 indicate that a program is an *embedded relational algebraic* entity as follows.

Theorem 1. The *Embedded Relational Model (ERM)* of programs states that a software system or a program \wp is a set of complex embedded relational processes, in which all previous processes of a given process form the context of the current process, i.e.:

$$\begin{aligned} \wp &= \mathbf{R}_{k=1}^m (@ e_k \hookrightarrow P_k) \\ &= \mathbf{R}_{k=1}^m [@ e_k \hookrightarrow \mathbf{R}_{i=1}^{n-1} (p_i(k) r_{ij}(k) p_j(k))], j = i+1, p_i \in \mathfrak{P}, r_{ij} \in \mathfrak{R} \end{aligned} \quad (23)$$

Proof. Theorem 1 can be directly proven on the basis of Definitions 17 and 21. Substituting P_k in Definition 21 with Eq. 17, a generic program \mathcal{P} obtains the form as a series of embedded relational processes as presented in Theorem 1.

The ERM model provides a unified mathematical treatment of programs, which reveals that a program is a finite nonempty set of embedded binary relations between a current statement and all previous ones that form the semantic context or environment of computing.

RTPA provides a coherent notation system and a formal engineering methodology for modeling both software and intelligent systems. RTPA can be used to describe both *logical* and *physical* models of systems, where logic views of the architecture of a software system and its operational platform can be described using the same set of

Table 3. Taxonomy of Denotational Mathematics

Operations		Concept Algebra	System Algebra	Real-Time Process Algebra			
				Meta-processes		Relational operations	
Relational operations	Related/independent	$\leftrightarrow / \leftrightarrow$	\sqsupset / \sqsubseteq	Assignment	$:=$	Sequence	\rightarrow
	Super/sub relation	\succ / \prec	$\leftrightarrow / \leftrightarrow$	Evaluation	\blacklozenge	Jump	\curvearrowright
	Equivalent	$=$	$=$	Addressing	\Rightarrow	Branch	$ $
	Consistent	\equiv		Memory allocation	\Leftarrow	Switch	$ \dots $
	Overlapped		Π	Memory release	\Leftarrow	While-loop	R^*
	Comparison	\sim		Read	\succ	Repeat-loop	R^+
	Definition	\triangleq		Write	\triangleleft	For-loop	R^i
Compositional operations	Inheritance	\Rightarrow	\Rightarrow	Input	$ \succ$	Recursion	\circ
	Tailoring	\Rightarrow	\Rightarrow	Output	$ \triangleleft$	Procedure call	\rightarrow
	Extension	$\overset{+}{\Rightarrow}$	$\overset{+}{\Rightarrow}$	Timing	$\textcircled{=}$	Parallel	\parallel
	Substitute	$\overset{\sim}{\Rightarrow}$	$\overset{\sim}{\Rightarrow}$	Duration	\triangleq	Concurrence	$\textcircled{\&}$
	Composition	\oplus	\oplus	Increase	\uparrow	Interleave	\parallel
	Decomposition	\uparrow	\uparrow	Decrease	\downarrow	Pipeline	\gg
	Aggregation/	\Leftarrow	\Leftarrow	Exception detection	$!$	Interrupt	\Leftarrow
	Specification	\vdash	\vdash	Skip	\circ	Time-driven dispatch	\hookrightarrow
	Instantiation	\mapsto	\mapsto	Stop	\boxtimes	Event-driven dispatch	\hookrightarrow
Difference		\ominus	System	\S	Interrupt-driven dispatch	\hookrightarrow	

notations. When the system architecture is formally modelled, the static and dynamic behaviors that perform on the system architectural model, can be specified by a three-level refinement scheme at the system, class, and object levels in a top-down approach. Detailed syntaxes and formal semantics of RTPA meta-processes and process relations may be referred to [37], [41], [43], [51], [52].

A summary of the algebraic operations and their notations in concept algebra, system algebra, and RTPA is provided in Table 3.

4 Applications of Denotational Mathematics

A wide range of applications of denotational mathematics has been identified, which encompass concept algebra for knowledge manipulations, system algebra for system architectural manipulations, and RTPA for system behavioral manipulations. This section presents some typical applications of denotational mathematics for the modeling of iterative and recursive systems architectures and behaviors by RTPA, the modeling of autonomous machine learning by concept algebra, and the modeling of granular computing by system algebra.

4.1 The Big-R Notation of RTPA for Modeling Iterative and Recursive System Architectures and Behaviors

The most generic and fundamental operations in system and human behavioral modeling are iterations and recursions. Because a variety of iterative constructs are provided in different programming languages, the notation for repetitive, cyclic, recursive behaviors and architectures in computing need to be unified.

The mechanism of the big-R notation can be analogized with the mathematical notations Σ or Π . To a great extent, Σ and Π can be treated as special cases of the big-R for repetitively doing additions and multiplications, respectively.

Example 1. The *big- Σ* notation $\sum_{i=1}^n x_i$ is a widely used calculus for denoting repetitive additions. Assuming that the operation of addition is represented by $sum(x)$, the mechanism of big- Σ can be expressed more generally by the big-R notation, i.e.:

$$\sum_{i=1}^n x_i = \mathbf{R}_{i=1}^n sum(x_i) \quad (24)$$

According to Definition 4, the big-R notation can be used to denote not only repetitive operational behaviors in computing, but also recurring constructs of architectures and data objects as shown below.

Example 2. The architecture of a two-dimensional array with $n \times m$ integer elements, A_{nm} , can be denoted by the big-R notation as follows:

$$A_{nm} = \mathbf{R}_{i=0}^{n-1} \mathbf{R}_{j=0}^{m-1} A[i, j] \mathbf{N} \quad (25)$$

Because the big-R notation provides a powerful and expressive means for denoting iterative and recursive behaviors, and architectures of systems or human beings, it is a universal mathematical means for system modeling in terms of repetitive behaviors and architectures, respectively.

Definition 22. An *infinite iteration* can be denoted by the big-R notation as:

$$\mathbf{R} P \triangleq \gamma \bullet P \curvearrowright \gamma \quad (26)$$

where γ is a label that denotes the fix (rewinding) point of a loop, and \curvearrowright denotes a jump in RTPA.

The infinite iteration may be used to formally describe an everlasting behavior of systems.

Example 3. A simple everlasting clock, *CLOCK*, which does nothing but tick as C.A.R. Hoare proposed [16], i.e.:

$$CLOCK \triangleq tick \rightarrow tick \rightarrow tick \rightarrow \quad (27)$$

can be efficiently denoted by the big-R notation as simply as follows:

$$CLOCK \triangleq \mathbf{R} tick \quad (28)$$

A more generic and useful iterative construct is the conditional iteration.

Definition 23. A *conditional iteration* can be denoted by the big-R notation as:

$$\begin{aligned} \mathbf{R}_{\text{exp}\mathbf{BL}=\mathbf{T}}^{\mathbf{F}} P \triangleq & \gamma \bullet (\blacklozenge \text{exp}\mathbf{BL} = \mathbf{T} \\ & \rightarrow P \\ & \curvearrowright \gamma \\ & | \blacklozenge \sim \\ & \rightarrow \emptyset \\ &) \end{aligned} \quad (29)$$

where \emptyset denotes a skip.

The conditional iteration is frequently used to formally describe repetitive behaviors on given conditions. Eq. 29 expresses that the iterative execution of P will go on as long as the evaluation \blacklozenge of the conditional expression is true ($\text{exp}\mathbf{BL} = \mathbf{T}$), until $\text{exp}\mathbf{BL} = \mathbf{F}$ abbreviated by ' \sim '.

Definition 24. *Recursion* $\mathbf{R}^{\circ} P^i$ is a multi-layered, embedded process relation in which a process P at layer i of embedment, P^i , calls itself at an inner layer $i-1$, P^{i-1} , $0 \leq i \leq n$. The termination of P^i depends on the termination of P^{i-1} during its execution, i.e.:

$$\begin{aligned}
\mathbf{R}^{\circ} P^i \triangleq & \mathbf{R}_{i\mathbf{N}=n\mathbf{N}}^0 (\blacklozenge i\mathbf{N} > 0 \\
& \rightarrow P^{\mathbf{N}} := P^{i\mathbf{N}-1} \\
& | \blacklozenge \sim \\
& \rightarrow P^0 \\
&)
\end{aligned} \tag{30}$$

where n is the *depth of recursion* or embedding that is determined by an explicitly specified conditional expression $\text{exp}\mathbf{BL} = \mathbf{T}$ inside the body of P .

Example 4. Using the big-R notation, the algorithm of the factorial function can be recursively defined as shown below:

$$\begin{aligned}
(n\mathbf{N})! \triangleq & \mathbf{R}^{\circ} (n\mathbf{N})! \\
= & \mathbf{R}_{i\mathbf{N}=n\mathbf{N}}^0 (\blacklozenge i\mathbf{N} > 0 \\
& \rightarrow (i\mathbf{N})! := i\mathbf{N} \bullet (i\mathbf{N}-1)! \\
& | \blacklozenge \sim \\
& \rightarrow (i\mathbf{N})! := 1 \\
&)
\end{aligned} \tag{31}$$

The big-R notation of RTPA captures a fundamental and widely applied mathematical concept in computing and human behavior description, which demonstrates that a convenient mathematical calculus and notation may dramatically reduce the difficulty and complexity in expressing a frequently used and highly recurring concept and notion in computing.

4.2 Autonomous Machine Learning Using Concept Algebra

Cognitive informatics [38], [39], [44] defines learning as a cognitive process at the higher cognitive function layer (Layer 7) according to the Layered Reference Model of the Brain (LRMB) [53]. The learning process is interacting with multiple fundamental cognitive processes such as *object identification*, *abstraction*, *search*, *concept establishment*, *comprehension*, *memorization*, and *retrievably testing*. Learning is closely related to other higher cognitive processes of inferences such as *deduction*, *induction*, *abduction*, *analogy*, *explanation*, *analysis*, *synthesis*, *creation*, *modeling*, and *problem solving*.

Definition 25. *Learning* is a higher cognitive process of the brain at the higher cognitive layer of LRMB that gains knowledge of something or acquires skills in some actions by updating the cognitive models in long-term memory.

According to the *Object-Attribute-Relation* (OAR) model [45], results of learning can be embodied by the updating of the existing OAR in the brain. In other words, learning is a dynamic composition of the currently created sub-OAR and the existing OAR in long-term memory (LTM) as expressed below.

Theorem 2. The *representation of learning result* states that the internal memory in the form of the OAR structure can be updated by a conjunction between the existing OAR and the newly created sub-OAR (sOAR), i.e.:

$$\begin{aligned} OAR' \mathbf{ST} &\triangleq OAR\mathbf{ST} \uplus sOAR\mathbf{ST} \\ &= OAR \mathbf{ST} \uplus (O_s, A_s, R_s) \end{aligned} \quad (32)$$

where the composition operation \uplus in concept algebra is defined below.

Definition 26. A *composition* of concept c from n subconcepts c_1, c_2, \dots, c_n , denoted by \uplus , is an integration of them that creates the new super concept c via concept conjunction, and establishes new associations between them, i.e.:

$$\begin{aligned} c(O, A, R^c, R^i, R^o) \uplus \prod_{i=1}^n c_i &\triangleq \\ c(O, A, R^c, R^i, R^o \mid O = \bigcup_{i=1}^n O_{c_i}, A = \bigcup_{i=1}^n A_{c_i}, \\ R^c = \bigcup_{i=1}^n (R_{c_i}^c \cup \{(c, c_i), (c_i, c)\}), R^i = \bigcup_{i=1}^n R_{c_i}^i, R^o = \bigcup_{i=1}^n R_{c_i}^o) \\ \parallel \prod_{i=1}^n c_i(O_i, A_i, R_{c_i}^c, R_{c_i}^i, R_{c_i}^o \mid R_{c_i}^i = R_{c_i}^i \cup \{(c, c_i)\}, R_{c_i}^o = R_{c_i}^o \cup \{(c_i, c)\}) \end{aligned} \quad (33)$$

As specified in Eq. 33, the composition operation results in the generation of new internal relations $\Delta R^c = \bigcup_{i=1}^n \{(c, c_i), (c_i, c)\}$ that is not belongs to any of its subconcepts. It is also noteworthy that, during learning by concept composition, the existing knowledge in forms of the individual n concepts is changed and updating concurrently via the newly created input/output relations with the newly generated concept.

Corollary 1. The *learning process* is a cognitive composition of a piece of newly acquired information and the existing knowledge in LTM in the form of the OAR-based knowledge networks.

The cognitive process of learning can be formally modeled using concept algebra and RTPA as given in Fig. 2. The center of the cognitive process of learning is that knowledge about the learn objects and intermediate results are represented internally in the brain as a sub-OAR model. According to the LRMB model [53] and the OAR model [45] of internal knowledge representation in the brain, the temporal result of learning in short-term memory (STM) is a new sub-OAR model, which will be used to update the entire OAR model of knowledge in LTM as permanent learning result.

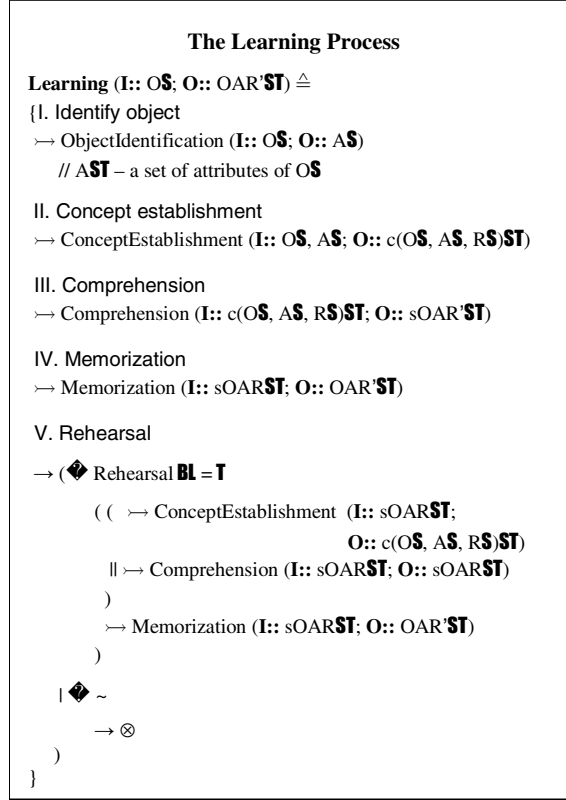


Fig. 2. Formal description of the learning process in concept algebra and RTPA

According to the formal model of the learning process, autonomic machine learning can be carried out by the following steps: 1) *Identify object*: This step identifies the learning object O ; 2) *Concept establishment*: This step establishes a concept model for the learning object O , $c(A, R, O)$, by searching related attributes A , relations R , and instances O ; 3) *Comprehension*: This step comprehends the concept and represents the concept with a sub-OAR model in STM; 4) *Memorization*: This step associates the learnt sub-OAR of the learning object with the entire OAR knowledge, and retains it in LTM; 5) *Rehearsal test*: This step checks if the learning result needs to be rehearsed. If yes, it continues to parallel execution of Steps (6) and (7); otherwise, it exits; 6) *Re-establishment of concept*: This step recalls the concept establishment process to rehearse the learning result; 7) *Re-comprehension*: This step recalls the comprehension process to rehearse the learning result.

The formalization of the cognitive process of learning does not only reveal the mechanisms of human learning, but also explain how machine may gain the capability of autonomous learning. Based on the rigorous syntaxes and semantics of RTPA, the formal learning process can be implemented by computers in order to form the core of machine intelligence [47].

4.3 Granular Computing Using System Algebra

The term *granule* is originated from Latin *granum*, i.e., grain, to denote a small compact particle in physics and in the natural world. The *taxonomy of granules* in computing can be classified into the data granule, information granule, concept granule, computing granule, cognitive granule, and system granule [20], [28], [33], [54], [58], [62].

Definition 27. A *computing granule*, shortly a *granule*, is a basic mathematical structure that possesses a stable topology and at least a unit of computational capability or behavior.

Definition 28. *Granular computing* is a new computational methodology that models and implements computational structures and functions by a granular system, where each granule in the system carries out a predefined function or behavior by interacting to other granules in the system.

It is recognized that any abstract or concrete granule can be formally modeled by abstract systems in system algebra. On the basis of Definition 13, an abstract granule can be formally described as follows.

Definition 29. A *computing granule* G on the *universal system environment* \mathcal{U} is a 7-tuple, i.e.:

$$G \triangleq S = (C, R^c, R^i, R^o, B, \Omega, \Theta) \quad (34)$$

where

- C is a finite nonempty set of *cell* or component of the system, $C = \{c_1, c_2, \dots, c_n\} \subseteq \mathcal{PC} \sqsubseteq \mathcal{U}$.
- R is a finite nonempty set of *relations* between pairs of the components in the system, $R = \{r_1, r_2, \dots, r_m\} \subseteq C \times C$.
- $R^c = C \times C$ is a set of *internal relations*.
- $R^i \subseteq C_\Theta \times C$ is a set of external *input relations*, $C_\Theta \subseteq \mathcal{PC} \sqsubseteq \mathcal{U}$.
- $R^o \subseteq C \times C_\Theta$ is a set of external *output relations*.
- B is a set of *behaviors* (or functions), $B = \{b_1, b_2, \dots, b_p\} \subseteq \mathcal{PB} \sqsubseteq \mathcal{U}$.
- Ω is a set of *constraints* on the memberships of components, the conditions of relations, and the scopes of behaviors, $\Omega = \{\omega_1, \omega_2, \dots, \omega_q\}$.
- Θ is the *environment* of G with a nonempty set of components C_Θ outside C , i.e., $\Theta = C_\Theta \subseteq \mathcal{PC} \sqsubseteq \mathcal{U}$.

Definition 30. A *granular system* S_G is a composition of multiple granules in a system where all granules interact with each other for a common goal of system functionality.

Properties of granular systems obey the properties of generic abstract systems as described in Section 3.2 [50]. The set of relational and compositional operations on granules and granular systems towards granular computing are identical as those modeled in system algebra.

5 Conclusions

The abstract, rigorous, and expressive needs in cognitive informatics, computational intelligence, software engineering, and knowledge engineering have led to new forms of mathematics collectively known as denotational mathematics. Denotational mathematics has been introduced as a category of expressive mathematical structures that deals with high-level mathematical entities such as abstract objects, complex relations, behavioral information, concepts, knowledge, processes, and systems. The domain and architecture of denotational mathematics have been described. New mathematical entities, novel mathematical structures, and applications of denotational mathematics have been explored toward the modeling and description of the natural and machine intelligent systems.

Extensions of conventional analytic mathematics onto more complicated mathematical entities beyond numbers and sets lead to the contemporary denotational mathematics. Three paradigms of denotational mathematics, such as concept algebra, system algebra, and Real-Time Process Algebra (RTPA), have been introduced. Within the new forms of denotational mathematics, *concept algebra* has been designed to deal with the new abstract mathematical structure of concepts and their representation and manipulation in knowledge engineering. RTPA has been developed to deal with series of behavioral processes and architectures of software and intelligent systems. *System algebra* has been created to the rigorous treatment of abstract systems and their algebraic relations and operations. Applications of denotational mathematics in cognitive informatics and computational intelligence have been elaborated with a set of case studies and examples. This work has demonstrated that denotational mathematics is an ideal mathematical means for a set of emerging disciplines that deal with concepts, knowledge, behavioral processes, and human/machine intelligence.

Acknowledgement. The author would like to acknowledge the Natural Science and Engineering Council of Canada (NSERC) for its partial support to this work. The author would like to thank the valuable comments and suggestions of the anonymous reviewers.

References

1. Aho, A.V., Sethi, R., Ullman, J.D.: *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Publication Co., New York (1985)
2. Anderson, J.R.: *The Architecture of Cognition*. Harvard Univ. Press, Cambridge (1983)
3. Ashby, W.R.: Principles of the Self-Organizing System. In: von Foerster, H., Zopf, G. (eds.) *Principles of Self-Organization*, pp. 255–278. Pergamon, Oxford (1962)
4. Ashby, W.R.: Requisite Variety and Implications for Control of Complex Systems. *Cybernetica* 1, 83–99 (1985)
5. Bender, E.A.: *Mathematical Methods in Artificial Intelligence*. IEEE CS Press, Los Alamitos (1996)
6. Chomski, N.: Three Models for the Description of Languages. *I.R.E. Transactions on Information Theory* 2(3), 113–124 (1956)

7. Codin, R., Missaoui, R., Alaoui, H.: Incremental Concept Formation Algorithms Based on Galois (Concept) Lattices. *Computational Intelligence* 11(2), 246–267 (1995)
8. Colins, A.M., Loftus, E.F.: A Spreading-Activation Theory of Semantic Memory. *Psychological Review* 82, 407–428 (1975)
9. Ellis, D.O., Fred, J.L.: *Systems Philosophy*. Prentice-Hall, Englewood Cliffs (1962)
10. Ford, J.: *Chaos: Solving the Unsolvable. Dynamics and Fractals*. Academic Press, London (1986)
11. Ganter, B., Wille, R.: *Formal Concept Analysis*, pp. 1–5. Springer, Heidelberg (1999)
12. Hampton, J.A.: Psychological Representation of Concepts of Memory, pp. 81–110. Psychology Press, Hove, England (1997)
13. Heylighen, F.: Self-Organization, Emergence and the Architecture of Complexity. In: Proc. 1st European Conf. on System Science (AFCET), Paris, pp. 23–32 (1989)
14. Higman, B.: *A Comparative Study of Programming Languages*, 2nd edn., MacDonald (1977)
15. Hoare, C.A.R.: Communicating Sequential Processes. *Communications of the ACM* 21(8), 666–677 (1978)
16. Hoare, C.A.R.: *Communicating Sequential Processes*. Prentice-Hall International, London (1985)
17. Hurlley, P.J.: *A Concise Introduction to Logic*, 6th edn. Wadsworth Pub. Co., ITP (1997)
18. Klir, G.J.: *Facets of Systems Science*. Plenum, New York (1992)
19. Lewis, H.R., Papadimitriou, C.H.: *Elements of the Theory of Computation*, 2nd edn. Prentice-Hall International, Englewood Cliffs (1998)
20. Lin, T.Y.: Granular Computing on Binary Relations (I): Data Mining and Neighborhood Systems. In: Proc. Rough Sets in Knowledge Discovery, pp. 107–120. Physica-Verlag, Heidelberg (1998)
21. Louden, K.C.: *Programming Languages: Principles and Practice*. PWS-Kent Publishing Co., Boston (1993)
22. Matlin, M.W.: *Cognition*, 4th edn. Harcourt Brace College Pub., NY (1998)
23. Medin, D.L., Shoben, E.J.: Context and Structure in Conceptual Combination. *Cognitive Psychology* 20, 158–190 (1988)
24. Milner, R. (ed.): *A Calculus of Communication Systems*. LNCS, vol. 92. Springer, Heidelberg (1980)
25. Murphy, G.L.: Theories and Concept Formation. In: Mechelen, I.V., et al. (eds.) *Categories and Concepts, Theoretical Views and Inductive Data Analysis*, pp. 173–200. Academic Press, London (1993)
26. Pavel, M.: *Fundamentals of Pattern Recognition*, 2nd edn. Addison-Wesley, Reading (1993)
27. Pawlak, Z.: Rough Logic, *Bulletin of the Polish Academy of Science. Technical Science* 5(6), 253–258 (1987)
28. Pedrycz, W. (ed.): *Granular Computing: An Emerging Paradigm*. Physica-Verlag, Heidelberg (2001)
29. Pnueli, A.: The Temporal Logic of Programs. In: Proc. 18th IEEE Symposium on Foundations of Computer Science, pp. 46–57. IEEE, Los Alamitos (1977)
30. O’Grady, W., Archibald, J.: *Contemporary Linguistic Analysis: An Introduction*, 4th edn. Pearson Education Canada Inc., Toronto (2000)
31. Rapoport, A.: Mathematical Aspects of General Systems Theory. *General Systems Yearbook* 11, 3–11 (1962)
32. Skarda, C.A., Freeman, W.J.: How Brains Make Chaos into Order. *Behavioral and Brain Sciences* 10 (1987)

33. Skowron, A., Stepaniuk, J.: Information Granules: Towards Foundations of Granular Computing. *International Journal of Intelligent Systems* 16, 57–85 (2001)
34. Smith, E.E., Medin, D.L.: *Categories and Concepts*. Harvard Univ. Press, Cambridge (1981)
35. von Bertalanffy, L.: *Problems of Life: An Evolution of Modern Biological and Scientific Thought*. C.A. Watts, London (1952)
36. von Neumann, J.: The Principles of Large-Scale Computing Machines. *Annals of History of Computers* 3(3), 263–273 (reprinted, 1946)
37. Wang, Y.: The Real-Time Process Algebra (RTPA). *Annals of Software Engineering: A International Journal, USA* 14, 235–274 (2002)
38. Wang, Y.: Keynote: On Cognitive Informatics. In: *Proc. 1st IEEE International Conference on Cognitive Informatics (ICCI 2002)*, Calgary, Canada, pp. 34–42. IEEE CS Press, Los Alamitos (2002)
39. Wang, Y.: On Cognitive Informatics. *Brain and Mind: A Transdisciplinary Journal of Neuroscience and Neurophilosophy, USA* 4(3), 151–167 (2003)
40. Wang, Y.: Using Process Algebra to Describe Human and Software System Behaviors. *Brain and Mind* 4(2), 199–213 (2003)
41. Wang, Y.: On the Informatics Laws and Deductive Semantics of Software. *IEEE Transactions on Systems, Man, and Cybernetics (C)* 36(2), 161–171 (2006)
42. Wang, Y.: Cognitive Informatics and Contemporary Mathematics for Knowledge Representation and Manipulation. In: Wang, G.-Y., Peters, J.F., Skowron, A., Yao, Y. (eds.) *RSKT 2006. LNCS (LNAI)*, vol. 4062, pp. 69–78. Springer, Heidelberg (2006)
43. Wang, Y.: *Software Engineering Foundations: A Software Science Perspective*. CRC Series in Software Engineering, vol. II. Auerbach Publications, USA (2007)
44. Wang, Y.: The Theoretical Framework of Cognitive Informatics. *International Journal of Cognitive Informatics and Natural Intelligence* 1(1), 1–27 (2007)
45. Wang, Y.: The OAR Model of Neural Informatics for Internal Knowledge Representation in the Brain. *International Journal of Cognitive Informatics and Natural Intelligence* 1(3), 64–75 (2007)
46. Wang, Y.: Keynote: On Theoretical Foundations of Software Engineering and Denotational Mathematics. In: *Proc. 5th Asian Workshop on Foundations of Software*, Xiamen, China, pp. 99–102 (2007)
47. Wang, Y.: The Theoretical Framework and Cognitive Process of Learning. In: *Proc. 6th International Conference on Cognitive Informatics (ICCI 2007)*, Lake Tahoe, CA, pp. 470–479. IEEE CS Press, Los Alamitos (2007)
48. Wang, Y.: On the Big-R Notation for Describing Iterative and Recursive Behaviors. *International Journal of Cognitive Informatics and Natural Intelligence* 2(1), 17–28 (2008)
49. Wang, Y.: On Concept Algebra: A Denotational Mathematical Structure for Knowledge and Software Modeling. *International Journal of Cognitive Informatics and Natural Intelligence* 2(2), 1–19 (2008)
50. Wang, Y.: On System Algebra: A Denotational Mathematical Structure for Abstract System Modeling. *International Journal of Cognitive Informatics and Natural Intelligence* 2(2), 20–42 (2008)
51. Wang, Y.: RTPA: A Denotational Mathematics for Manipulating Intelligent and Computational Behaviors. *International Journal of Cognitive Informatics and Natural Intelligence* 2(2), 44–62 (2008)
52. Wang, Y.: Deductive Semantics of RTPA. *International Journal of Cognitive Informatics and Natural Intelligence* 2(2), 95–121 (2008)

53. Wang, Y., Wang, Y., Patel, S., Patel, D.: A Layered Reference Model of the Brain (LRMB). *IEEE Trans. on Systems, Man, and Cybernetics (C)* 36(2), 124–133 (2006)
54. Wang, Y., Lotfi, A.: On the System Algebra Foundation for Granular Computing. *International Journal of Software Science and Computational Intelligence* 1(1) (December 2008)
55. Wille, R.: Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts. In: Rival, I. (ed.) *Ordered Sets*, pp. 445–470. Reidel, Dordrecht (1982)
56. Wilson, L.B., Clark, R.G.: *Comparative Programming Language*, Wokingham, UK. Addison-Wesley Publishing Co., Reading (1988)
57. Woodcock, J., Davies, J.: *Using Z: Specification, Refinement, and Proof*. Prentice Hall International, London (1996)
58. Yao, Y.Y.: Information Granulation and Rough Set Approximation. *International Journal of Intelligent Systems* 16(1), 87–104 (2001)
59. Yao, Y.Y.: A Comparative Study of Formal Concept Analysis and Rough Set Theory in Data Analysis. In: *Proc. Rough Sets and Current Trends in Computing*, pp. 59–68 (2004)
60. Zadeh, L.A.: Fuzzy Sets and Systems. In: Fox, J. (ed.) *Systems Theory*, pp. 29–37. Polytechnic Press, Brooklyn (1965)
61. Zadeh, L.A.: Outline of a New Approach to Analysis of Complex Systems. *IEEE Trans. on Sys. Man and Cyb.* 1(1), 28–44 (1973)
62. Zadeh, L.A.: Fuzzy Sets and Information Granularity. In: Gupta, M.M., Ragade, R., Yager, R. (eds.) *Advances in Fuzzy Set Theory and Applications*, pp. 3–18. North-Holland, Amsterdam (1979)