ON FORMAL AND COGNITIVE SEMANTICS FOR SEMANTIC COMPUTING

YINGXU WANG

Theoretical and Empirical Software Engineering Research Centre (TESERC)
Department of Electrical and Computer Engineering
Schulich School of Engineering, University of Calgary
2500 University Drive, NW, Calgary
Alberta, Canada, T2N 1N4
yingxu@ucalgary.ca

Semantics is the meaning of symbols, notations, concepts, functions, and behaviors, as well as their relations that can be deduced onto a set of predefined entities and/or known concepts. Semantic computing is an emerging computational methodology that models and implements computational structures and behaviors at semantic or knowledge level beyond that of symbolic data. In semantic computing, formal semantics can be classified into the categories of to be, to have, and to do semantics. This paper presents a comprehensive survey of formal and cognitive semantics for semantic computing in the fields of computational linguistics, software science, computational intelligence, cognitive computing, and denotational mathematics. A set of novel formal semantics, such as deductive semantics, concept-algebra-based semantics, and visual semantics, is introduced that forms a theoretical and cognitive foundation for semantic computing. Applications of formal semantics in semantic computing are presented in case studies on semantic cognition of natural languages, semantic analyses of computing behaviors, behavioral semantics of human cognitive processes, and visual semantic algebra for image and visual object manipulations.

Keywords: Semantic computing; formal semantics; deductive semantics; concept-algebra-based semantics; visual semantics; cognitive semantics; relational semantics; behavioral semantics; cognitive computing; cognitive informatics; denotational mathematics; computing with words.

1. Introduction

Semantic computing is an emerging computing methodology and technology that was initiated by Philip Sheu and his colleagues in 2006 [54, 55], who perceived semantic computing as “computing with machine processable descriptions of content and intentions.” Semantic computing brings together multiple disciplines concerned with connecting the often vaguely formulated intentions of humans with computational content that includes, but is not limited to, structured and semi-structured data, multimedia data, text, programs, services and, even, network behavior [54]. Semantic computing is cross fertilized by multimedia computing [55], semantic webs
1 Semantic computing extends conventional symbolic and syntax-driven computing technologies to semantic and knowledge processing.

Definition 1. **Semantics** is the meaning of symbols, notations, concepts, functions, and behaviors, as well as their relations that can be deduced onto a set of predefined entities and/or known concepts.

Semantic analysis and comprehension are a deductive cognitive process. According to the Object-Attribute-Relation (OAR) model for internal knowledge representation [64], the semantics of a sentence in a natural language may be considered to be understood when: (a) The logical relations of parts of the sentence are clarified; and (b) All parts of a sentence are reduced to the terminal entities, which are either a real-world image or a primitive abstract concept.

Semantics can be classified into the categories of *to be*, *to have*, and *to do* semantics as shown in Table 1 [63]. A “to be” semantics infers the meaning of an equivalent relation between an unknown and known entities or concepts. A “to have” semantics provides the meaning of a structure or a composite entity. A “to do” semantics provides the meaning of an action or behavior of a system or a human.

Semantics may also be classified into formal and informal semantics. The former are those defined and manipulated in mathematics or formal notations; the latter are those described and conveyed in natural languages. The result of a cognition of a given abstract object is a *to be* semantics; and that of the consequence of a behavior is a *to do* semantics.

Definition 2. **Computing**, in a narrow sense, is an application of computers to solve a given computational problem by imperative instructions; while in a broad sense, it is a process to implement the instructive intelligence by a system that transfers a set of given information or instructions into expected behaviors.

On the basis of the analyses of the concepts of semantics and computing, the intention and extension of semantic computing may be described as follows.

Definition 3. **Semantic computing** is an emerging computational methodology that models and implements computational structures and behaviors at semantic or knowledge level beyond that of symbolic data.

<table>
<thead>
<tr>
<th>Category</th>
<th>Semantic objects</th>
<th>Formal means for semantic computing</th>
</tr>
</thead>
<tbody>
<tr>
<td>To be</td>
<td>Relations</td>
<td>Logic, concept algebra [66], granular algebra [71]</td>
</tr>
<tr>
<td>To have</td>
<td>Structures</td>
<td>Set, system algebra [74]</td>
</tr>
<tr>
<td>To do</td>
<td>Behaviors</td>
<td>Function, process algebra (RTPA [60])</td>
</tr>
</tbody>
</table>
This paper presents a comprehensive survey of the formal and cognitive semantics for semantic computing. Theories and applications of various formal semantics are described in the fields of computational linguistics, software science, computational intelligence, cognitive computing, and denotational mathematics. Section 1 elaborates formal semantics in computational linguistics. Section 2 reviews formal semantics in computing and software science. Section 3 discusses formal semantics in cognitive computing and computational intelligence. Section 4 presents applications of formal semantics in semantic computing. A set of new formal semantics, such as deductive semantics, concept-algebra-based semantics, and visual semantics, is introduced in this paper toward developing the theoretical and cognitive foundations of semantic computing.

2. Formal Semantics in Computational Linguistics

Studies on formal semantics are initiated in linguistics and natural language processing, which can be traced back to Taski (1944) and Chomski (1956). This section elaborates formal semantics of natural languages and its relations with syntaxes. Wang’s deductive semantics [62] for natural language processing is introduced. A concept-algebra-based technology [66] for semantic modelling and analysis is developed for semantic computing and computing with words [82]. Then, a comparative study on fundamental theories of natural and programming languages is presented by contrasting their syntaxes, semantics, and grammars.

2.1. The deductive semantics of natural languages

Semantics is a domain of linguistics that studies the interpretation of words and sentences, and analyses their meanings. In linguistics, semantics deals with how the meaning of a sentence in a language is obtained and comprehended. Studies on semantics explore mechanisms in the understanding of language and the nature of meaning where syntactic structures play an important role in the interpretation of sentence and the intension and extension of word meaning [9–14, 59, 65].

**Definition 4.** The *mathematical model of semantics* of natural languages, $\mathcal{S}$, is a 5-tuple, i.e.:

$$
\mathcal{S} \triangleq (J, B, O, T, S)\tag{1}
$$

where

- $J$ is the subject of the sentence
- $B$ is a behavior or action
- $O$ is the object of the sentence
- $T$ is the time when the action is occurring
- $S$ is the space where the action is occurring

According to Definition 4 ($\mathcal{S}$), the relationship between a language and its syntaxes and semantics can be illustrated as shown in Fig. 1. Which explains that
linguistic analyses are a deductive process that maps the 1-D language into the 5-D semantics via the 2-D syntactical analyses.

**Theorem 1.** The semantics of a sentence in a natural language is comprehended iff:
1. The logical relations of parts of the sentence are clarified;
2. All parts of sentence are reduced to the terminal entities, which are either a real-world image or a primitive abstract concept.

Semantic analysis and comprehension are a deductive cognition process. Further discussions on the theoretical foundations of language cognition and comprehension can be found in [9, 24, 59, 62, 63, 66].

### 2.2. Semantic analyses by concept algebra

Concepts are the basic unit of both knowledge and thinking [2, 16, 20, 27, 32, 39, 44, 62, 66]. Therefore, rigorous modeling and formal treatment of concepts are at the center of theories for knowledge presentation and manipulation [15, 44, 56, 79]. A concept in linguistics is a noun or noun-phrase that serves as the subject of a to-be statement [32, 63]. A new denotational mathematic structure known as concept algebra [66] is introduced in this subsection for a formal treatment of abstract concepts.

#### 2.2.1. The mathematical model of formal concepts

**Definition 5.** A concept is a cognitive unit to identify and/or model a real-world concrete entity and a perceived-world abstract object.

Before an abstract concept is defined, the semantic environment or context [20, 27, 32, 41, 66] in a given language, is introduced.

**Definition 6.** 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 $\Theta$ is denoted as
a triple, i.e.:

$$\Theta \triangleq (O, A, R)$$

where $R$ is a set of relations between $O$ and $A$, and $|$ denotes alternative relations.

Concepts in denotational mathematics [65] are an abstract structure that carries certain meaning in almost all cognitive processes such as thinking, learning, and reasoning.

**Definition 7.** An abstract concept $c$ on the semantic environment $\Theta$ is a 5-tuple, i.e.:

$$c \triangleq (O, A, R^c, R^i, R^o)$$

where $O$ is a nonempty set of objects of the concept, $O = \{a_1, a_2, \ldots, a_m\} \subseteq P^O(PO$ denotes a power set of $O$); $A$ is a nonempty set of attributes, $A = \{a_1, a_2, \ldots, a_n\} \subseteq P.A; R^c = O \times A$ is a set of internal relations; $R^i \subseteq A' \times A, A' \subseteq C' \wedge A \subseteq c,$ is a set of input relations, where $C'$ is a set of external concepts, $C' \subseteq \Theta.$ For convenience, $R^i = A' \times A$ may be simply denoted as $R^i = C' \times c$; and $R^o \subseteq c \times C'$ is a set of output relations.

### 2.2.2. Concept algebra for semantic analyses and manipulations

Concept algebra, developed by Wang in 2008 [66], is an abstract mathematical structure for the formal treatment of concepts and their algebraic relations, operations, and associative rules for composing complex concepts.

**Definition 8.** A concept algebra $CA$ on a given semantic environment $\Theta$ is a triple, i.e.:

$$CA \triangleq (C, OP, \Theta) = ([O, A, R^c, R^i, R^o], \{•_{r}, •_{c}\}, \Theta)$$

where $OP = \{•_{r}, •_{c}\}$ are the sets of relational and compositional operations on abstract concepts.

**Definition 9.** The relational operations $•_{r}$ in concept algebra encompass 8 comparative operators for manipulating the algebraic relations between concepts, i.e.:

$$•_{r} \triangleq \{\leftrightarrow, \not\leftrightarrow, \prec, \not\prec, =, \not=, \sim, \not\sim\}$$

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

**Definition 10.** The compositional operations $•_{c}$ in concept algebra encompass 9 associative operators for manipulating the algebraic compositions among
concepts, i.e.:

\[
\circlearrowleft \triangleq \{ \Rightarrow, \not\Rightarrow, \Rightarrow, \cup, \subseteq, \triangleright, \rightarrow, \}
\]

(6)

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

Detailed descriptions of the relational and compositional operations of concept algebra have been rigorously defined in [66]. Concept algebra provides a denotational mathematical means for algebraic manipulations of abstract concepts in semantic computing. 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, cognitive computing, software engineering, and system engineering.

2.3. Comparative analysis of natural and programming language theories

On the basis of the formal semantics, a comparative study can be conducted between the linguistic properties of natural and programming language, where the fundamental expressiveness of natural languages can be classified as shown in Table 2. It is observed [63] that although natural languages can be rich, complex, and powerfully descriptive, they do share three common basic structures known as the meta-expressivenesses of ‘to be (|=),’ ‘to have ( |=),’ and ‘to do (>|)’ as shown in Table 2.

A comparative analysis of natural and programming languages is summarized in Table 3. Intuitively, it was expected that a programming language would be a small subset of natural languages. Surprisingly, this hypothesis is only partially true at the morphology (lexicon) and semantic levels. However, the syntax of programming languages is far more complicated than those of natural languages.

It is noteworthy in Table 3 that the semantics of programming languages is much simpler than that of natural languages, which is determined by the basic objectives of applications that should be suitable for limited machine intelligence. However, for achieving such simple and precise semantics in programming languages, a set of very complex and rigorous syntax and grammatical rules has to be adopted. Further discussion on semantics of programming languages may be referred to [1, 35, 40, 58, 62, 63, 67, 68].

More generally, there is no clear-cut between syntax and semantics in both natural and programming languages as formally stated below.

<table>
<thead>
<tr>
<th>Function</th>
<th>Category</th>
<th>Notation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify objects and attributes</td>
<td>To be</td>
<td></td>
<td>=</td>
</tr>
<tr>
<td>Describe relations and possession</td>
<td>To have</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Describe status and behaviors</td>
<td>To do</td>
<td></td>
<td>&gt;</td>
</tr>
</tbody>
</table>
On Formal and Cognitive Semantics for Semantic Computing

Table 3. Comparative analysis of natural and programming language properties.

<table>
<thead>
<tr>
<th>No</th>
<th>Category</th>
<th>Natural language</th>
<th>Programming language</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Phonetics</td>
<td>Small</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>Phonology</td>
<td>Complex</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>Morphology (lexis)</td>
<td>Very large (&gt;60,000 words)</td>
<td>Small (&lt;1,000 instructions/ reserved words)</td>
</tr>
<tr>
<td>4</td>
<td>Syntax</td>
<td>Simple (&lt;100 rules)</td>
<td>Very complicated (&gt;1,000 rules)</td>
</tr>
<tr>
<td>5</td>
<td>Semantics</td>
<td>Very complex (5-D)</td>
<td>Simple (2-D)</td>
</tr>
<tr>
<td>6</td>
<td>Grammar</td>
<td>Context sensitive</td>
<td>Context free</td>
</tr>
<tr>
<td>7</td>
<td>Applications</td>
<td>Thought, communications</td>
<td>Computing, system control</td>
</tr>
</tbody>
</table>

Theorem 2. The principle of tradeoff between syntaxes and semantics states that in formal linguistics, the complexities of the syntactic rules (or grammar) $C_{syn}$ and of the semantic rules $C_{sem}$ are inversely proportional, i.e.:

$$C_{syn} \propto \frac{1}{C_{sem}}$$

Theorem 2 indicates that the simpler the syntactic rules or the grammar, the richer or complicated the semantics, and vice versa. Because natural languages are relatively more simple, its semantics are much richer, complicated, and more ambiguous. On the contrary, because programming languages adopt very detailed and complicated grammars, their semantics are relatively concise, simple, and rigorous. The finding in Theorem 2 indicates that syntactic and semantic rules are equivalent and interchangeable in linguistics. A simple syntax will be required for a complex semantics, while a complex syntax will result in a simple semantics.

3. Formal Semantics in Computing and Software Science

Studies on software semantics have been recognized as one of the key areas in the development of fundamental theories for computer science and software science [25, 28, 40]. The semantics of a programming language is mainly classified as a “to do” semantics, which is the behavioral meanings that constitute what a syntactically correct instructional statement in the language is supposed “to do” during runtime. The development of formal semantic theories of programming is one of the pinnacles of computing and software engineering [6, 29, 30, 36, 63].

In semantics analyses, the instructions shared by all programming languages can be classified into three types: (a) Internal operations such as memory manipulation and assignment instructions; (b) Basic control structures such as the if-then-else and while-do instructions; and (c) External operations onto the environment, such as input/output, event handling, and human-machine interactive instructions.

Definition 11. The semantics of a program in a given programming language is the logical consequences of an execution of the program that results in the changes
of values of a finite set of variables and/or the embodiment of computing behaviors in the underlying computing environment.

This section introduces existing semantic theories of programming languages such as target semantics, operational semantics, denotational semantics, axiomatic semantics, and algebraic semantics. Then, it demonstrates how the semantics of a program in a given programming language is expressed and embodied. It also shows that the existing semantic notations and methodologies are inadequate to express some important instructions, complex control structures, and the real-time environments at run-time. This leads to the development of the deductive semantics as described in Sec. 3.2.

3.1. **Taxonomy of computational semantics**

Basic semantics of a programming language can be described by its *behavioral equivalence* to another known language, such as a natural language or languages of the target machines. Semantics can also be described by a set of predefined executable functions in machine languages. Another approach to specify the semantics of a programming language is by mathematical definitions known as formal semantics.

A number of formal semantics, such as the *operational* [37, 77, 78], *denotational* [7, 33, 48–52], *axiomatic* [18, 19, 23, 28], *algebraic* [21, 22, 26], and *deductive semantics* [62, 67], have been proposed in the last three decades for defining and interpreting the meanings of program behaviors.

3.1.1. **Target semantics**

The most basic and simplest semantics of programming languages is the target semantics, which maps the equivalent behaviors of a given statement in the target-machine’s language. Target semantics is typical semantics adopted during early stages of programming technologies.

**Definition 12.** Target semantics is an equivalent semantics that adopts a target-machine’s language to interpret the behavioral meaning of a program in a programming language.

The most typical target language is assembly language. However, there are two drawbacks in using target semantics because machine languages are system dependent. The first drawback is that it is not rigorous and cannot be formally defined and described in order to facilitate machine-based semantic analyses. Another is the low efficiency in applications because the semantics of a given programming language has to be mapped into multiple target languages. These reasons motivated the studies on theories of formal semantics of programming languages and software systems.

A common approach towards the establishment of formal semantics is to develop suitable and generic abstract models of the target machines, supplemented with the formal description of the abstract syntactic rules of the programming languages. Once the target machines can be abstracted by unified mathematic models, formal
semantics such as the operational, denotational, axiomatic, and algebraic semantics may be developed as shown in the following subsections.

3.1.2. Operational semantics

Definition 13. Operational semantics adopts a virtual machine, whose operations are well defined, to describe the semantics of a program in a specific programming language by its equivalent behaviors implemented on the virtual machine.

The foundation of operational semantics is based on virtual machine theory [40]. A typical virtual machine for embodying operational semantics of an arbitrary program is called a reduction machine [36]. The reduction machine is used to reduce the given program to values inside the machine and its environment by a finite set of permissible operations.

3.1.3. Denotational semantics

Definition 14. Denotational semantics adopts functions to describe the semantics of a programming language, in which the functions map semantic values into syntactically legal program constructs.

The foundation of denotational semantics is based on recursive function theory [7, 33, 48–52]. Denotational semantics is considered as a well defined semantics among existing ones for expressing the meaning of computational instructions in a programming language. In denotational semantics, instructions in a program can be translated into a set of functions based on rigorously defined methodologies.

3.1.4. Axiomatic semantics

Definition 15. Axiomatic semantics adopts effective assertions to describe the semantics of a programming language, in which the assertions of effects for executing an instruction are deduced to the values of data objects manipulated by the instruction.

The foundation of axiomatic semantics is based on predicate logic [18, 19, 23, 28], where assertions play an important role in axiomatic semantics. Because logical axioms are used in the assertions for denoting program semantics, this method gains the name axiomatic semantics.

Definition 16. An assertion $A$ is a logical statement about the predicate behavior $Q$ and its initial assumptions $P$ of a given instruction $S$ at any given point of a program during run-time, which can be examined as true or false, i.e.:

$$A \triangleq \{P\} S \{Q\}$$

(8)

where $P$ is called the precondition and $Q$ the postcondition.
Assertions play an important role in correctness proving for formal specifications of software systems. Nevertheless, assertions have been adopted in a number of modern programming languages, such as C++ and Java. However, \( P \) and \( Q \) are not always specifiable, because some of the operations at run-time are unpredictable or indeterministic, such as event dispatching, parallel mechanisms, and dynamic memory allocations.

### 3.1.5. Algebraic semantics

**Definition 17.** Algebraic semantics adopts abstract algebra to describe the semantics of a programming language, in which data objects and operations are defined by algebraic axioms and deduced by abstract algebraic laws.

The foundation of algebraic semantics is based on abstract algebras [21, 22, 26]. A well-known application of algebraic semantics is the definitions and descriptions of Abstract Data Types (ADTs). Algebraic semantics are capable to deduce the semantics of data objects and imposed operations on abstract types, sorts, and mathematical entities. However, it cannot reduce the semantics onto concrete data entities and complex architectures and processes.

### 3.1.6. Deductive semantics

The aforementioned classic formal semantics were oriented on a certain set of software behaviors that are limited by the models of their semantic environments. The mathematical models of the target machines and the semantic environments in conventional semantics seem to be inadequate to deal with the semantics of complex programming requirements, and to express some important instructions, complex control structures, and the real-time environments at run-time. For supporting systematic and machine enabled semantic analysis and code generation in software engineering, the deductive semantics [62, 67] will be introduced that provides a systematic semantic analysis methodology.

### 3.2. Deductive semantics

Deduction is an inference process that discovers new knowledge or derives a specific conclusion based on generic premises such as abstract rules or principles. The nature of semantics of a given programming language is its computational meanings or embodied behaviors expressed by an instruction in the language. Because the carriers of software semantics are a finite set of variables declared in a given program, program semantics can be reduced onto the changes of values of these variables over time. In order to provide a rigorous mathematical treatment of both the abstract and concrete semantics of software, a new type of formal semantics known as the deductive semantics is developed [62, 67].

**Definition 18.** Deductive semantics is a formal software semantics that deduces the semantics of a program in a given programming language from a generic
abstract semantic function to the concrete semantics, which are embodied onto the
changes of status of a finite set of variables constituting the semantic environment
of computing.

The theoretical foundations of deductive semantics are based on process algebra
and Boolean partial differentials [62, 63]. Based on the mathematical models
and architectural properties of a program at different composing levels, deductive
models of software semantics, semantic environment, and semantic matrix will be
formally defined. Properties of software semantics and relations between the soft-
ware behavioral space and semantic environment will be elaborated in Sec. 3.3. A
comprehensive analysis of the deductive semantics of Real-Time Process Algebra
(RTPA) may be referred to [58, 60, 63].

Deductive semantics can be used to define both the abstract and concrete
semantics of large-scale software systems, facilitate software comprehension and
recognition, support tool development, enable semantics-based software testing and
verification, and explore the semantic complexity of software systems.

This subsection develops the mathematical models of deductive semantics. The
deductive models of semantics, semantic function, and semantic environment at var-
ious composing levels of programs are introduced. Properties of software semantics
and relationships between the software behavioral space and the semantic envi-
ronment are studied. New methods such as the semantic differential and seman-
tic matrix are developed to facilitate deductive semantic analyses from a generic
semantic function to a specific semantic matrix, and from semantics of statements to
those of processes and programs. The establishment of the deductive semantic rules
of RTPA may be referred to [67], where deductive semantics of a comprehensive set
of software processes is modeled.

3.2.1. The mathematical model of software semantics

A semantic environment of a program in a given programming language is a logi-
cal model of a finite set of identifiers and their values changing over time along the
execution of the program. The semantic environment constituting behaviors of soft-
ware is inherently a three-dimensional structure known as the operations, memory
space, and time.

**Definition 19.** The behavioral space $\Omega$ of a program executed on a certain
machine is a finite set of variables operated in a 3-D state space determined by
a triple, i.e.:

$$\Omega \triangleq (OP, T, S)$$

where $OP$ is a finite set of operations, $T$ is a finite set of discrete steps of program
execution, and $S$ is a finite set of memory locations or their logical representations by
identifiers of variables. The set of variables of a program, $S$, plays an important role
In semantic analysis, because they are the objects of software behavioral operations and the carriers of program semantics.

On the basis of the definitions of software behavioral space and partial differential of sets [63], the semantic environment of software can be introduced as follows.

**Definition 20.** The semantic environment $\Theta$ of a program on a certain target machine is its run-time behavioral space $\Omega(OP,T,S)$ projected onto the Cartesian plane determined by $T$ and $S$, i.e.:

$$
\Theta = \frac{\partial^2 \Omega}{\partial t \partial s}, \quad t \in T \land s \in S
$$

$$
= \frac{\partial^2 \Omega}{\partial t \partial s}(OP,T,S) = T \times S
$$

where $T$ is a finite set of discrete steps of program execution, $S$ is a finite set of memory locations or their logical representations by identifiers of variables.

As indicated in Definition 20, the semantic environment of a program is a dynamic entity over time, because following each execution of a statement in the program, the semantic environment $\Theta$, particularly the set of values $V$ of the variables $S$ may be changed as a result of the operation of the statement.

A generic semantic function is developed below, which can be used to derive a specific and concrete semantic function for a given statement, process, or program at different composing levels by mathematical deduction.

**Definition 21.** A semantic function of a program $\wp$, $f_{\wp}(\varphi)$, is a function that maps the semantic environment $\Theta$ into a finite set of values $V$ determined by a Cartesian product on a finite set of executing steps $T$ and a finite set of variables $S$, i.e.:

$$
f_{\wp}(\varphi) = f : T \times S \to V
$$

$$
= \begin{pmatrix}
    s_1 & s_2 & \cdots & s_m \\
    t_0 & \bot & \bot & \cdots & \bot \\
    t_1 & v_{11} & v_{12} & \cdots & v_{1m} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    t_n & v_{n1} & v_{n2} & \cdots & v_{nm}
\end{pmatrix}
$$

(11)

where $T = \{t_0, t_1, \ldots, t_n\}$, $S = \{s_1, s_2, \ldots, s_m\}$, and $V$ is a finite set of values $v(t_i, s_j)$, $0 \leq i \leq n$, and $0 \leq j \leq m$.

In Eq. (11), all values of $v(t_i, s_j)$ at $t_0$ are undefined for a program as denoted by the bottom symbol $\bot$, i.e., $v(0, s_j) = \bot$, $1 \leq j \leq m$. However, for a statement or a process, it is usually true that $v(0, s_j) \neq \bot$ dependent on the context of previous statement(s) or the initialization of the system.
According to Definition 21, the semantic environment and the domain of a semantic function can be illustrated by a semantic diagram [67] as described below.

**Definition 22.** A semantic diagram is a sub-Cartesian-plane in the semantic environment $\Theta$ that forms the domain of the semantic function for a composed process $P$ with $f_\Theta(P) = f : T_P \times S_P \rightarrow V_P$.

The semantic diagram $f_\Theta(P)$ as modeled in Definition 22 can be illustrated in Fig. 2, where $S_P$ is the set of variables of process $P$.

The semantic diagram can be used to analyze complex semantic relations, and to demonstrate semantic functions and their semantic environments. Observing Fig. 2, the flowing properties of semantic function for composed processes can be derived.

**Theorem 3.** The variables of two arbitrary processes $P$ and $Q$, $S_P$ and $S_Q$, in the semantic environment $\Theta$ possess the following properties:

(a) The entire set of variables: $S = S_P \cup S_Q$

(b) Global variables: $S_G \subseteq S_P \cap S_Q$

(c) Local variables: $S_L = S - S_G$, $S_L \subseteq S_P \oplus S_Q$, where $S_{L_P} = S_L \setminus S_Q$ and $S_{L_Q} = S_L \setminus S_P$

3.2.2. Deductive semantics of programs at different levels of compositions

It is noteworthy that deductive semantics introduces only a universal abstract semantic function as given in Definition 21 rather than adopting multiple concrete semantic functions as conventional approaches do. In deductive semantics, any particular concrete semantic function is a deduced instantiation of the universal abstract semantic function. This is why it is named deductive semantics, and this avoids the trouble in other exhaustive approaches where a new semantic function has to be particularly defined from time to time whenever additional instruction is introduced in a given language.

![Fig. 2. The semantic diagram of a process.](image)
The semantics of a program in a given language can be described and analyzed at various composition levels, known as those of statement, process, and system from the bottom up.

Definition 23. The semantics of a statement \( p, \theta(p) \), on a given semantic environment \( \Theta \) is a double partial differential of the semantic function \( f_\theta(p) \) on the sets of variables \( S \) and executing steps \( T \), i.e.:

\[
\theta(p) = \frac{\partial^2}{\partial t \partial s} f_\theta(p) = \sum_{i=0}^{\#T(p)} \sum_{j=1}^{\#S(p)} v_p(t_i, s_j)
\]

\[
= \sum_{i=0}^{1} \sum_{j=1}^{\#(s_1, s_2, \ldots, s_m)} v_p(t_i, s_j)
\]

\[
= \begin{pmatrix} v_{p(t_0, s_1)} & v_{p(t_0, s_2)} & \cdots & v_{p(t_0, s_m)} \\
\end{pmatrix}
\]

(15)

where \( t \) denotes the discrete time immediately before and after the execution of \( p \) during \( [t_0, t_1] \), and \( \# \) is the cardinal calculus that counts the number of elements in a given set, i.e., \( n = \#T(p) \) and \( m = \#S(p) \).

In Definition 23, the first partial differential selects all related variable \( S(p) \) of the statement \( p \), and the second partial differential selects a set of discrete steps of \( p \)'s execution \( T(p) \) from \( \Theta \). According to Definition 22, the semantics of a statement can be deduced onto a semantic function that results in a 2-D matrix with the changes of values of all variables over time of program execution.

In semantic analysis, the changed part of the semantic environment \( \Theta \), known as the semantic effect as defined below, is particularly interested, which is the embodiment of software semantics.

Definition 24. The semantic effect of a statement \( p, \theta^*(p) \), is the resulting changes of values of variables by its semantic function \( \theta(p) \) during the time interval immediately before and after the execution of \( p \), \( \Delta t = [t_i, t_{i+1}] \), i.e.:

\[
\theta^*(p) = \sum_{j=1}^{\#S(p)} (v_p(t_i, s_j) \oplus v_p(t_{i+1}, s_j))
\]

\[
= \sum_{j=1}^{\#S(p)} (v_p(t_i, s_j) \rightarrow v_p(t_{i+1}, s_j)) | v_p(t_i, s_j) \neq v_p(t_{i+1}, s_j)
\]

(16)

where \( \rightarrow \) denotes a transition of values for a given variable.

Because a program or a process is composed by individual statements with given rules of compositions, the definitions and mathematical models of semantics...
at the statement level can be extended onto the higher levels of program hierarchy systematically.

**Definition 25.** The *semantics of a process* $P$, $\theta(P)$, on a given semantic environment $\Theta$ is a double partial differential of the semantic function $f_\theta(P)$ on the sets of variables $S$ and executing steps $T$, i.e.:

$$
\theta(P) = \frac{\partial^2}{\partial t \partial s} f_\theta(P)
$$

$$
= \sum_{k=1}^{n-1} R_k \left\{ \frac{\partial^2}{\partial t \partial s} f_\theta(P_k) \right\} r_{kl} \left\{ \frac{\partial^2}{\partial t \partial s} f_\theta(P_l) \right\}, \quad l = k + 1
$$

$$
= \sum_{k=1}^{n-1} \left[ \#T(P_k) \#S(P_k) \right] R_{i=0} R_{j=1} V_{P_k}(t_i, s_j) \left[ \#T(P_l) \#S(P_l) \right] R_{i=0} R_{j=1} V_{P_l}(t_i, s_j)
$$

$$
= \begin{pmatrix}
V_{P_1} & V_G \\
V_{P_2} & V_G \\
\vdots & \vdots \\
V_{P_{n-1}} & V_G
\end{pmatrix}
$$

(17)

where $V_{P_k}, 1 \leq k \leq n - 1$, is a set of values of local variables that belongs to processes $P_k$, and $V_G$ is a finite set of values of global variables.

On the basis of Definition 25, the semantics of a program at the top-level composition can be deduced to the combination of semantics of a set of processes, each of which can be further deduced to the composition of all statements’ semantics as described below.

**Definition 26.** The *semantics of a program* $\varphi$, $\theta(\varphi)$, on a given semantic environment $\Theta$, is a combination of the semantic functions of all processes $\theta(P_k), 1 \leq k \leq n$, i.e.:

$$
\theta(\varphi) = \sum_{k=1}^{#K(\varphi)} R_k \frac{\partial^2}{\partial t \partial s} f_\theta(\varphi)
$$

$$
= \sum_{k=1}^{#K(\varphi)} \theta(P_k)
$$

$$
= \sum_{k=1}^{#K(\varphi)} \left[ \#T(P_k) \#S(P_k) \right] R_{i=0} R_{j=1} V_{P_k}(t_i, s_j)
$$

(18)

where $#K(\varphi)$ is the number of processes or components in the program.

It is noteworthy that Eq. (18) will usually result in a very large matrix of semantic space, which can be quantitatively predicated as follows.
Definition 27. The *semantic space of a program* $S_\Theta(\varphi)$ is a product between the number of variables $\#S(\varphi)$ and the number of executing steps $\#T(\varphi)$, i.e.:

$$S_\Theta(\varphi) = \#S(\varphi) \cdot \#T(\varphi) = \sum_{k=1}^{\#K(\varphi)} \#S(\varphi_k) \cdot \sum_{k=1}^{\#K(\varphi)} \#T(\varphi_k).$$  

(19)

The semantic space of a program provides a useful measure for software complexity. Due to the tremendous size of the semantic space, both program composition and comprehension are innately a hard problem in terms of complexity and cognitive difficulty.

3.3. **Properties of behavioral semantics of software**

Observing the formal definitions and mathematical models of deductive semantics developed in preceding subsections, a number of common properties of software semantics may be elicited, which are useful for explaining the fundamental characteristics of software semantics [63].

One of the most interesting characteristics of program semantics is its invariance against different executing speeds as described in the following theorem.

**Theorem 4.** The asynchronicity of program semantics states that the semantics of a relatively timed program is invariant with the changes of executing speed, as long as any absolute time constraint is met.

The above theorem states that, for most nonreal-time or relatively timed programs, different executing speeds or simulation paces will not alter the semantics of the software system. This explains why a programmer may simulate the run-time behaviors of a given program that may be executing at a speed of up to $10^9$ times faster than that of human beings. It also explains why computers with different system clock frequencies may correctly run the same program and obtain the same behavior.

A fundamental question in programming language and software engineering theories is what the *least complete set of instructions* for programming is. Based on the sets of sufficient metainstructions and their algebraic compositional rules [63], the questions regarding the least complete set of instructions in programming can be formally answered below.

**Theorem 5.** The least complete set of instructions in programming states that a program is composable with sufficient descriptive power in a given language iff both the sufficient sets of metainstructions ($\mathcal{P}$) and compositional rules ($\mathcal{R}$) are rigorously defined.
Theorem 5 indicates that the necessary and sufficient conditions of program compositionality in a given language are that all the metainstructions (P) and the fundamental BCS’s (R) must be implemented in the language.

Definition 28. The behavior of a computational statement is a set of observable actions or changes of status of objects operated by the statement.

According to Definitions 19 and 20, the behavioral space of software $\Omega$ is three-dimensional, while the semantic environment $\Theta$ is two-dimensional. Therefore, to a certain extent, semantic analysis is a projection of the 3-D software behaviors into the 2-D semantic environment $\Theta$ as shown in Fig. 3.

4. Formal Semantics in Computational Intelligence

Machine understandable semantics of visual and multimedia information is the third category of formal semantics. A variety of theories and approaches are proposed for visual object and pattern recognition, particularly the machine cognition of visual semantics. Marr proposed a method for object recognition in the algorithmic approach known as the computational method [38]. Biederman developed a method for object recognition in the analytic approach called recognition by components [5]. Various methods and technologies are developed for pattern recognition in the fields of cognitive psychology [46, 47, 80], computer science [3, 8, 43, 57], and robotics [31, 45]. Wang presents a cognitive theory of visual information processing as well as the unified framework of human visual processing systems [73] in the development of cognitive informatics — a formal theory for explaining the natural and computational intelligence [61, 64, 75, 76].

This section presents the cognitive process of visual semantic analyses and their denotational mathematical means [65]. In order to efficiently model the abstract visual objects, their semantic representations, and their rigorous compositions and manipulations, a new denotational mathematics known as Visual Semantic Algebra (VSA) [73] is introduced in this section for pattern recognition and visual semantic analyses.
4.1. Visual semantics of images and patterns

The basic geometric shapes (2-D) and solids (3-D), known collectively as geons, have been studied in cognitive psychology [5], computational intelligence, and robotics [31, 45]. A set of 26 typical visual semantic objects are summarized in Table 4, in four categories known as those of plane geometry, solid geometry, generic figures,

<table>
<thead>
<tr>
<th>No.</th>
<th>Category</th>
<th>Basic Geom</th>
<th>Symbol</th>
<th>Mathematical model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Plane geometry (H)</td>
<td>Point</td>
<td>$\cdot \triangle$</td>
<td>$(x, y)$</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Line</td>
<td>$\cdot \triangle$</td>
<td>$(x_1, y_1), (x_2, y_2)$, $l = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Angle</td>
<td>$\leq$</td>
<td>$((x, y), (x_0, y_0), (x_0, y_1), (x_0, y_2))$</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Arc</td>
<td>$\cdot \triangle$</td>
<td>$(x, y), t, (a_1, a_2)$</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Triangle</td>
<td>$\cdot \triangle$</td>
<td>$(x, y, (x_1, y_1), (x_2, y_2))$</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Square</td>
<td>$\parallel$</td>
<td>$(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$, $</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Rectangle</td>
<td>$\parallel$</td>
<td>$(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$, $</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Parallelogram</td>
<td>$\parallel$</td>
<td>$(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$, $</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>Rhomb</td>
<td>$\parallel$</td>
<td>$(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$, $</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Circle</td>
<td>$\cdot \triangle$</td>
<td>$(x, y), r$</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>Ellipse</td>
<td>$\cdot \triangle$</td>
<td>$(x, y), (x_0, y_0), (x_0, y_1),</td>
</tr>
<tr>
<td>12</td>
<td>Solid geometry (S)</td>
<td>Cube</td>
<td>$C_2 \cdot \triangle$</td>
<td>$(x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3), (x_4, y_4, z_4), (x_5, y_5, z_5), (x_6, y_6, z_6), (x_7, y_7, z_7), (x_8, y_8, z_8), (x_9, y_9, z_9)$, $</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>Rectangular solid</td>
<td>$R_2 \cdot \triangle$</td>
<td>$(x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3), (x_4, y_4, z_4), (x_5, y_5, z_5), (x_6, y_6, z_6), (x_7, y_7, z_7), (x_8, y_8, z_8), (x_9, y_9, z_9)$, $</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>Cylinder</td>
<td>$C_2 \cdot \triangle$</td>
<td>$(x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3), (x_4, y_4, z_4), (x_5, y_5, z_5), (x_6, y_6, z_6), (x_7, y_7, z_7), (x_8, y_8, z_8), (x_9, y_9, z_9)$, $</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>Sphere</td>
<td>$S_2 \cdot \triangle$</td>
<td>$(x, y, z)$, $r$, $\sqrt{x_1^2 + y_1^2 + z_1^2} = r$</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>Cone</td>
<td>$C_2 \cdot \triangle$</td>
<td>$(x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3), (x_4, y_4, z_4), (x_5, y_5, z_5), (x_6, y_6, z_6), (x_7, y_7, z_7), (x_8, y_8, z_8), (x_9, y_9, z_9)$, $</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>Pyramid</td>
<td>$P_2 \cdot \triangle$</td>
<td>$(x_1, y_1, z_1), (x_2, y_2, z_2)$, $(x_3, y_3, z_3), (x_4, y_4, z_4), (x_5, y_5, z_5), (x_6, y_6, z_6)$, $h =</td>
</tr>
<tr>
<td>18</td>
<td>Generic figures (F)</td>
<td>Abstract human</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>Abstract system/machine</td>
<td>$\uparrow$</td>
<td>$\uparrow$</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>Abstract object</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>21</td>
<td>Abstract spatial limits (L)</td>
<td>Ceiling</td>
<td>$\Uparrow$</td>
<td>$\Uparrow$</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>Bottom (Ground)</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>Left limit</td>
<td>$\leftarrow$</td>
<td>$\leftarrow$</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>Right limit</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>Back limit</td>
<td>$\leftarrow$</td>
<td>$\leftarrow$</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>Front limit</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
</tr>
</tbody>
</table>
and abstract spatial limits. In Table 4, each abstract visual object is rigorously defined as given in the corresponding mathematical model.

**Definition 29.** The abstract visual objects O, or geons, in VSA are a set of 26 basic 2-D shapes \( H \), 3-D solids \( S \), generic figures \( F \), and abstract spatial limits \( L \) as summarized in Table 4, i.e.:

\[
O \triangleq \{ H \cup S \cup F \cup L \} \tag{20}
\]

where the four categories of abstract objects can be identified as follows, respectively:

\[
H \triangleq \{ \bullet, -, \angle, \triangle, \Box, \square, \diamond, \circ \} \tag{21}
\]

\[
S \triangleq \{ C_u, R_s, C_y, S_p, C_o, P_y \} \tag{22}
\]

\[
F \triangleq \{ \uparrow, \downarrow, \leftarrow, \rightarrow, \odot, \otimes, \sqcap \} \tag{23}
\]

\[
L \triangleq \{ \top, \bot, \vdash, \dashv, \llbracket, \rrbracket \} \tag{24}
\]

### 4.2. Visual semantic algebra

**Definition 30.** Visual Semantic Algebra (VSA) is a denotational mathematical structure that formally manipulates visual objects by algebraic operations on symbolic or semantic objects in geometric analyses and compositions, i.e.:

\[
\text{VSA} \triangleq (O, \bullet_{\text{VSA}}) \tag{25}
\]

where \( O \) is a finite set of basic abstract visual objects and \( \bullet_{\text{VSA}} \) is a finite set of algebraic operations on \( O \).

A set of 13 algebraic operations, as described in Table 5, is elicited from relational compositions of the 26 abstract visual objects. In Table 5, any 2-D or 3-D geometric structure can be analyzed or composed semantically using VSA.

**Definition 31.** The algebraic operations, \( \bullet_{\text{VSA}} \), in VSA are a set of 13 fundamental relational operations on the basic abstract visual objects \( O \) as summarized in Table 5, i.e.:

\[
\bullet_{\text{VSA}} \triangleq \left\{ \uparrow, \downarrow, \leftarrow, \rightarrow, \odot, \otimes, \angle, @, @((p), @((x,y,x), \land, \rightarrow, \bigwedge_{i=0}^{n-1} A_i \mapsto A_{i+1}) \right\}. \tag{26}
\]

VSA provides a new paradigm of denotational mathematical means for relational visual object manipulation and semantic analyses. VSA may be applied not only in machine visual and spatial reasoning, but also in computational intelligent system design as a powerful man-machine language in representing and dealing with visual inferences in complex geometrical and pattern systems. On the basis of VSA, computational intelligence systems such as robots and cognitive computers can process and inference visual and image objects rigorously and efficiently at the conceptual and semantic levels.
Table 5. Algebraic operations on abstract visual objects in VSA.

<table>
<thead>
<tr>
<th>Relational operations</th>
<th>Symbol</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Above</td>
<td>↑</td>
<td>$S_1 \uparrow S_2$</td>
<td>$S_1$ is above $S_2$.</td>
</tr>
<tr>
<td>Below</td>
<td>↓</td>
<td>$S_1 \downarrow S_2$</td>
<td>$S_1$ is below $S_2$.</td>
</tr>
<tr>
<td>Left</td>
<td>←</td>
<td>$S_1 \leftarrow S_2$</td>
<td>$S_1$ is on the left of $S_2$.</td>
</tr>
<tr>
<td>Right</td>
<td>→</td>
<td>$S_1 \rightarrow S_2$</td>
<td>$S_1$ is on the right of $S_2$.</td>
</tr>
<tr>
<td>Front</td>
<td>⊙</td>
<td>$S_1 \odot S_2$</td>
<td>$S_1$ is in front of $S_2$.</td>
</tr>
<tr>
<td>Behind</td>
<td>⊗</td>
<td>$S_1 \otimes S_2$</td>
<td>$S_1$ is behind $S_2$.</td>
</tr>
<tr>
<td>Inside</td>
<td>⊞</td>
<td>$S_1 \oslash S_2$</td>
<td>$S_1$ is inside $S_2$.</td>
</tr>
<tr>
<td>Angle</td>
<td>∠$[x^0]$</td>
<td>$S_1 \angle [x^0] S_2$</td>
<td>$S_1$ is at an $x^0$ angle position related to $S_2$ ($0^\circ$ is defined at the right position).</td>
</tr>
<tr>
<td>Relative position</td>
<td>@(p)</td>
<td>$S @(p)$</td>
<td>$S$ is allocated at the position $p$.</td>
</tr>
<tr>
<td>Absolute position</td>
<td>@(x, y, z)</td>
<td>$S @(x, y, z)$</td>
<td>$S$ is allocated at the position $(x, y, z)$.</td>
</tr>
<tr>
<td>Move</td>
<td>↢</td>
<td>$S @(p_1) ↢ @(p_2)$</td>
<td>$S$ moves from position $p_1$ to $p_2$.</td>
</tr>
<tr>
<td>Action (Sequential)</td>
<td>→</td>
<td>$S : (Act_1 \rightarrow Act_2)$</td>
<td>$S$ executes action$1$ then action$2$.</td>
</tr>
<tr>
<td>Action (Repetitive)</td>
<td>$R^{N-1}<em>{i=0} (A_f \rightarrow A</em>{i+1})$</td>
<td>$S : R^{N-1}<em>{i=0} (A_f \rightarrow A</em>{i+1})$</td>
<td>$S$ executes a set of actions, $A_i, 0 \leq i \leq n$, in a sequence.</td>
</tr>
</tbody>
</table>

5. Applications of Formal Semantics and Denotational Mathematics in Semantic Computing

A wide variety of applications of semantic computing has been identified, which extends computational structures and behaviors from conventional symbolic data level to the semantic or knowledge level. This section describes applications of the formal semantic theories and denotational mathematics for semantic computing. Case studies and examples are provided in semantics cognition of natural languages, semantic analyses of computing behaviors, behavioral semantics of human cognitive processes, and visual semantic algebra for Image and visual object manipulations.

5.1. Semantics cognition of natural languages

Formal semantics and concept algebra provide linguists, particularly language analyzers, implementers, and recognizers, for a powerful tool to formally describe and process natural language documents. Perspective applications of deductive semantics are identified in the development of Internet searching engines, semantic analysis of natural languages, speech recognitions, and intelligent systems for natural language parsing and knowledge processing.

Although concept itself is a dynamic and interlinked entity in knowledge, a higher level of human knowledge may be modeled by the interrelations and...
interactions among individual concepts. Such knowledge can be modeled by a formal concept network [66] and illustrated by a concept graph [66], which connects related concepts by relations. Complex interrelation between concept groups may be described as combinations of the fundamental relations modeled in $\mathcal{R} = \bullet_c$, especially concept compositions [66]. For example, complex concept relations such as causality and behaviors can be modeled as derived relations using concept algebra.

**Definition 32.** A concept network $CN$ is a hierarchical network of concepts interlinked by the set of nine associations $\mathcal{R}$ defined in concept algebra, i.e.:

$$CN = \mathcal{R} : \prod_{i=1}^{n} C_i \rightarrow \prod_{i=1}^{n} C_i$$  \hspace{1cm} (27)

A fundamental concept network skeleton is constructed on the basis of the general cognitive taxonomy WordNet (1993) [81], which reflect basic concept interrelation and organization of human knowledge. The nine compositional operations defined in concept algebra can be used to formally model WordNet to form different facets of the concept network.

A semantic concept network formed by compositional operations of concept algebra can be illustrated by a concept graph as given in Example 1.

**Example 1.** An abstract concept network that is formed by the composition and aggregation of a set of related concepts $c_0$ through $c_9$, as well as objects $o_1$ through $o_3$, can be illustrated by a concept graph as shown in Fig. 4.

![Concept Graph](image)

Fig. 4. An abstract concept network.
The formal description corresponding to the above concept network can be
carried out using concept algebra as given below.

\[
c_0 \uparrow \!
\big((c_1 \Rightarrow c_3 \rightarrow c_6 \rightarrow o_1) \parallel (c_1 \Rightarrow c_4 \rightarrow o_2)\big) \\
\parallel \!
\big((c_2 \vdash c_5 \nvdash (c_7 \parallel (c_8 \rightarrow o_3))\big) \\
c_9 \leftarrow (c_6 || c_7) = ((c_1 \Rightarrow c_3 \rightarrow c_6) || (c_2 \vdash c_5 \nvdash c_7))
\]

(28)

The case study in Example 1 demonstrates that concept algebra and concept
network are a generic and formal semantic knowledge manipulation means, which
are capable of dealing with complicated abstract or concrete knowledge structures
as well as their algebraic semantics and operations in semantic computing.

5.2. Semantic analyses of computing behaviors

On the basis of Definitions 21 and 23 as given in Sec. 3.2, the semantics of any
statements in a given programming language can be analyzed using the generic
model of semantic functions (Eq. (11)) via a deductive process.

Example 2. Analyze the semantics of Statement 3, \( p_3 \), in the following program
entitled \textit{sum}.

```c
void sum;
{
(0) int x, y, z;
(1) x = 8;
(2) y = 2;
(3) z := x + y;
}
```

According to Definition 21, the semantics of Statement \( p_3 \) is as follows:

\[
\theta(p_3) = \frac{\partial^2}{\partial t \partial s} f_\theta(p_1) \\
= \sum_{i=2}^{3} R_{i} \ sum_{j=1}^{R_{i}} v_{p_3}(t_i, s_j) \\
= \sum_{i=2}^{3} \#\{x,y,z\} \sum_{j=1}^{R_{i}} v_{p_3}(t_i, s_j) \\
= \begin{pmatrix} x & y & z \\ t_2 & 8 & 2 & \bot \\ (t_2, t_3) & 8 & 2 & 10 \end{pmatrix}.
\]

(29)

This example shows how the concrete semantics of a statement can be derived
on the basis of the generic and abstract semantic function of deductive semantics.
Example 3. For the same statement $p_3$ as shown in Example 2, determine its semantic effect $\theta^*(p_3)$.

According to Eq. (16), the semantic effect $\theta^*(p_3)$ is:

$$
\theta^*(p_3) = \#(S(p_3)) = \frac{1}{R_j \sum_i (v_{p_3}(t, s_j) \rightarrow v_{p_3}(t, s_j))}
$$

$$
= \#(x, y, z) \frac{1}{R_j \sum_i (v_{p_3}(t, s_j) \rightarrow v_{p_3}(t, s_j))}
$$

$$
= \{v_{p_3}(t, z) = \bot \rightarrow v_{p_3}(t, z) = 10\}.
$$

It can be seen in Examples 2 and 3 that deductive semantics can be used not only to describe the abstract and concrete semantics of programs, but also to elicit and highlight their semantic effects.

Deductive semantics can also be applied in semantic analyses of the metaprocesses and process relations of RTPA [67] as a comprehensive denotational mathematical means for program modeling in semantic computing.

Example 4. The semantics of the assignment process, $\theta(y_{RT} := x_{RT})$, in the given semantic environment $\Theta$ is a double partial differential of the semantic function $f_0(y_{RT} := x_{RT})$ on the sets of variables $S$ and executing steps $T$, i.e.:

$$
\theta(y_{RT} := x_{RT}) \triangleq \frac{\partial^2}{\partial t \partial s} f_0(y_{RT} := x_{RT})
$$

$$
= \frac{\#T(y_{RT} := x_{RT}) \#S(y_{RT} := x_{RT})}{\sum_i R_i \sum_j R_j (v(t, s_j))}
$$

$$
= \frac{1}{R_i \sum_j R_j} v(t, s_j)
$$

$$
= \begin{pmatrix}
  x_{RT} & y_{RT} \\
  t_0 & x_{RT} & \bot \\
  (t_0, t_1) & x_{RT} & x_{RT}
\end{pmatrix}
$$

(30)

where the size of the matrix is $\#T \bullet \#S$.

Example 5. The semantics of the sequential relations of processes, $\theta(P \rightarrow Q)$, in the given semantic environment $\Theta$ is a double partial differential of the semantic function $f_0(P \rightarrow Q)$ on the sets of variables $S$ and executing steps $T$, i.e.:

$$
\theta(P \rightarrow Q) \triangleq \frac{\partial^2}{\partial t \partial s} f_0(P \rightarrow Q)
$$

$$
= \frac{\partial^2}{\partial t \partial s} f_0(P) \rightarrow \frac{\partial^2}{\partial t \partial s} f_0(Q)$$
where $PQ$ indicates a concatenation of these two processes over time, and in the
simplified notation of the matrix, $V_P = v(t_P, S_P), 0 \leq t_P \leq n_P, 1 \leq S_P \leq m_P$;
$V_Q = v(t_Q, S_Q), 0 \leq t_Q \leq n_Q, 1 \leq S_Q \leq m_Q$; and $V_{PQ} = v(t_{PQ}, S_{PQ}), 0 \leq t_{PQ} \leq n_{PQ}, 1 \leq S_{PQ} \leq m_{PQ}$.

In Eq. (31), the first partial differential selects a set of related variables in the
sequential processes $P$ and $Q, S(P \cup Q)$. The second partial differential selects a set
of time moments $T(PQ)$. The semantic diagram of the sequential process relation
as defined in Eq. (31) is illustrated in Fig. 5 in the semantic environment $\Theta$.

The following example shows the physical meaning of Eq. (31) and how the
abstract syntaxes and their implied meanings are embodied onto the objects
(variables) and their dynamic values in order to obtain the concrete semantics in
deductive semantics.
**Example 6.** Analyze the semantics of two simple sequential processes \( P \) and \( Q \) in the following program:

```c
void sequential_sum
{
    (0) int x, y, z;
    { // P
        (1) x = 2;
        (2) y = 8;
        (3) z := x + y;
    }
    { // Q
        (4) z := x + y + z;
        (5) print z;
    }
}
```

According to Definition 21 and Example 5, the semantics of the above program can be analyzed as follows:

\[
\theta(P_0 \rightarrow P_1 \rightarrow \cdots \rightarrow P_5) = \left. \frac{\partial^2}{\partial t \partial s} f_\theta(P_0 \rightarrow P_1 \rightarrow \cdots \rightarrow P_5) \middle| \right. \\
= \left. \frac{\partial^2}{\partial t \partial s} f_\theta(P_0) \right| \rightarrow \frac{\partial^2}{\partial t \partial s} f_\theta(P_1) \rightarrow \cdots \rightarrow \frac{\partial^2}{\partial t \partial s} f_\theta(P_5) \\
= R_i^#T(P_i) \cdot R_j^#S(P_i) \cdot \#T(P_i) \cdot \#S(P_i) \\
\rightarrow R_i^#T(P_i) \cdot R_j^#S(P_i) \cdot \#T(P_i) \cdot \#S(P_i) \\
= R_i^#T(P_i) \cdot R_j^#S(P_i) \cdot v(t_i, s_j) \\
= 5 \cdot 4 \cdot v(t_i, s_j)
\]

\[
\begin{pmatrix}
    x & y & z & PORT(CRT)P|M \\
    t_0 & \bot & \bot & \bot \\
    (t_0, t_1) & 2 & \bot & \bot \\
    (t_1, t_2) & 2 & 8 & \bot \\
    (t_2, t_3) & 2 & 8 & 10 \\
    (t_3, t_4) & 2 & 8 & 20 \\
    (t_4, t_5) & 2 & 8 & 20
\end{pmatrix}
\]
where \( \text{PORT}[\text{CRT}] \| \) denotes a system monitor of type \( \mathbb{N} \) located by the pointer \( \text{CRT}P \); the semantics of \( P \) and \( Q \) are shown in the intervals \([t_0, t_3]\) and \((t_3, t_5)\), respectively.

**Example 7.** The semantics of the while-loop relations of processes, \( \theta(R_{\text{expBL}=T}^* (P)) \), in the given semantic environment \( \Theta \) is a double partial differential of the semantic function \( f_\theta(R_{\text{expBL}=T}^* (P)) \) on the sets of variables \( S \) and executing steps \( T \), i.e.:

\[
\theta \left( R_{\text{expBL}=T}^* (P) \right) \triangleq \frac{\partial^2}{\partial t \partial S} f_\theta \left( R_{\text{expBL}=T}^* (P) \right)
\]

\[
= R_{\text{expBL}=T}^* \left( \frac{\partial^2}{\partial t \partial S} f_\theta (P) \right)
\]

\[
= R_{\text{expBL}=T}^* \left( \#T(P) \#S(P) \right)
\]

\[
= R_{\text{expBL}=T}^* \left( \sum_{i=0}^{\#T(P)} \sum_{j=1}^{\#S(P)} v_p(t_i, S_j) \right)
\]

\[
= \begin{pmatrix}
\text{expBL} & S_p \\
[t_0, t_1] & \delta(\text{expBL}) \perp \\
(t_1, t_2) & T & V_p \\
(t_1, t_2) & F \otimes \\
& & \vdots \\
(t_3, t_4) & \delta(\text{expBL}) \dashv \\
(t_4, t_5) & T & V_p \\
(t_4, t_5) & F \otimes \\
\end{pmatrix}
\]

where \( \emptyset \) denotes exit, and \( \delta(\text{expBL}) \) is the evaluation function on the Boolean expression, \( \delta(\text{expBL}) \in \{T, F\} \).

The semantic diagram of the while-loop process relation as defined in Eq. (33) is illustrated in Fig. 6 in the semantic environment \( \Theta \).

Fig. 6. The semantic diagram of the while-loop process relation.
The theory of deductive semantics developed in Sec. 3.2 can be systematically applied to formally and rigorously describe the semantics of the RTPA metaprocesses and the process relations (operations) [60, 63]. Because RTPA is a mathematical modeling language based on process algebra that covers a comprehensive set of computing and programming requirements, any formal semantics that is capable of processing RTPA is powerful enough to express the semantics of any programming language in semantic computing.

5.3. Behavioral semantics of human cognitive processes

In cognitive informatics [64, 69, 70, 76], learning is defined as a cognitive process at the higher cognitive function layer according to the Layered Reference Model of the Brain (LRMB) [64]. The learning process interacts 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, analysis, synthesis, and problem solving [64].

Definition 33. 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 (LTM).

According to the Object-Attribute-Relation (OAR) model [64], results of learning can be embodied by the updating of the existing OAR in the brain as a concept network. In other words, learning is a dynamic composition of the currently created sub-OAR and the existing OAR in LTM.

Definition 34. A composition of concept $c$ from $n$ subconcepts $c_1, c_2, \ldots, c_n$, 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.:

$$c(O, A, R^c, R^i, R^0) \uplus \bigoplus_{i=1}^{n} R_{c_i} \triangleq \bigcup_{i=1}^{n} \left( R_{c_i} \cup \{(c, c_i), (c_i, c)\} \right)$$

$$R^c = \bigcup_{i=1}^{n} \left( R_{c_i} \cup \{(c, c_i), (c_i, c)\} \right)$$

$$R^i = \bigcup_{i=1}^{n} \left( R_{c_i} \cup \{(c, c_i), (c_i, c)\} \right)$$

$$R^0 = \bigcup_{i=1}^{n} \left( R_{c_i} \cup \{(c, c_i), (c_i, c)\} \right)$$

As specified in Eq. (34), 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 does not belong to any of its subconcepts. It is also noteworthy that, during learning by concept composition, the existing knowledge in the forms of individual $n$ concepts is concurrently changed and updated via the newly created input/output relations with the newly generated concept.
Example 8. 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. 7. 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 and the OAR model 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.

According to the formal model of the learning process, autonomic machine learning can be carried out in semantic computing 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 by concept algebra and RTPA does not only reveal the mechanisms of human learning, but also explain how machine may gain the capability of autonomic learning in semantic computing. 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 [64].

### 5.4. Visual semantic algebra for image and visual object manipulations

As described in Sec. 4.2, VSA provides a neat and powerful algebraic system for rigorously manipulating visual objects and patterns. Any 2D or 3D visual structure or system can be analyzed or composed using VSA. Therefore, VSA enables robots and intelligent systems to process visual semantic information and spatial behaviors in semantic computing.

**Example 9.** The visual pattern of the solid structures of $A$ and $B$ as given in Fig. 8 can be expressed in VSA as follows:

$$A \triangleq C_o \uparrow C_y \uparrow @ (\text{center}) R_s \uparrow \bot$$

$$B \triangleq P_y \uparrow @ (\text{center}) R_s \uparrow (C_{y1} \leftarrow C_{y2})$$

where $(o) \uparrow \bot$ denotes that the object $o$ is on the top of the ground.

**Example 10.** A more complex case to deal with a robot walking down stairs (Fig. 9) can be formally described in VSA. The visual walk planning mechanisms

![Fig. 8. Compositions of solids in VSA.](image)
Algorithm 3. A Robot Walks Down Stairs

\begin{verbatim}
WDS_Algorithm ≜ WDS_Architecture$ \leftrightarrow WDS_Behavior ↑

WDS_Architecture$ (\text{IF}) ≜

{ // Layout
      \text{StairST} : (S_1 \text{ST} \uparrow \uparrow (S_2 \text{ST} \uparrow \uparrow (S_3 \text{ST} \uparrow \uparrow (\ldots (S_n \text{ST} \uparrow \perp \ldots)))))

// Robot
      \text{Robot ↑} : (\text{LeftFootST} \parallel \text{RightFootST})

// Initial state
      \text{Robot ↑} @ (\text{← StairST}_1 \text{ST}, \text{ST}, \perp)

// Final state
      \text{Robot ↑} @ (\text{← StairST}_1, \text{ST}, \perp)

}

WDS_Behavior ↑ ≜ WDS(\text{Robot ↑}, WDS_Architecture$, \text{IF}) ::

\{ 
  \text{R} (\text{IF}) = 0
  \rightarrow \text{Robot ↑} . \text{LeftFootST} \uparrow \perp \rightarrow (\text{StairST}_1 \text{ST}, \text{ST}, \perp)
  \rightarrow \text{Robot ↑} . \text{RightFootST} \uparrow \perp \rightarrow (\text{StairST}_1 \text{ST}, \text{ST}, \perp)
  \rightarrow \text{Robot ↑} . \text{LeftFootST} \uparrow \perp \rightarrow (\text{StairST}_1 \text{ST}, \text{ST}, \perp)
  \rightarrow \text{Robot ↑} . \text{RightFootST} \uparrow \perp \rightarrow (\text{StairST}_1 \text{ST}, \text{ST}, \perp)
\}
\end{verbatim}

Fig. 9. A robot walks down stairs (Honda ASIMO, from Wikipedia).

Fig. 10. The algorithm of WDS in VSA.
and processes can be described by the walking down stairs (WDS) algorithm as given in Fig. 10.

Algorithm 1. The algorithm of WDS algorithm can be described in VSA as shown in Fig. 10. The WDS_Algocomplexes the architecture WDS_Architecture§, the robot behaviors WDS_Behaviors †, and their interactions. WDS_Architecture§ describes the layout, initial and final states of the system. WDS_Behaviors † describes the actions of the robot based on its visual interpretation about the stairs’ visual structures.

The theory and case studies presented in this subsection demonstrate that VSA provides a new paradigm of denotational mathematical means for relational visual object manipulation in semantic computing. VSA can be applied not only in machine visual and spatial reasoning, but also in computational intelligence system designs as a powerful man-machine language in representing and dealing with the high-level inferences in complex visual patterns and systems. On the basis of VSA, computational intelligence systems such as robots and cognitive computers can process and reason visual and image objects and their spatial relations rigorously and efficiently at conceptual level in semantic computing.

6. Conclusions

This paper has surveyed the theoretical foundations and technical approaches of formal and cognitive semantics for semantic computing. Various existing formal semantics in the fields of computational linguistics, software science, computational intelligence, cognitive computing, and denotational mathematics are reviewed. A set of novel formal semantics newly developed in cognitive computing and denotational mathematics has been introduced such as deductive semantics, conceptual-algebra-based semantics, and visual semantics toward establishing the theoretical and cognitive foundations of semantic computing.

It has been recognized that semantics is the meaning of symbols, notations, concepts, functions, and behaviors, as well as their relations that can be deduced onto a set of predefined entities and/or known concepts. Formal semantics in the denotational mathematical forms have been classified into the categories of to be (relational), to have (structural), and to do (behavioral) semantics. A wide variety of applications of formal semantic theories in semantic computing has been identified and demonstrated in this paper, such as semantics cognition of natural languages, semantic analyses of computing behaviors, behavioral semantics of human cognitive processes, and visual semantic algebra for image and visual object manipulations.

Acknowledgments

The author would like to acknowledge the Natural Science and Engineering Council of Canada (NSERC) for its partial support to this work. I would like to thank the anonymous reviewers for their valuable suggestions and comments on this work.
I sincerely thank Prof. Philip Sheu, for his vision and leadership on semantic computing and his comments on the earlier versions of this paper. I also express my thanks to the organizers of IEEE WSCE’08 for an invited keynote on formal semantics related to this article.

References


[38] D. Marr, *Vision* (Freeman, San Francisco, 1982).


