Deductive Semantics of RTPA

Yingxu Wang, University of Calgary, Canada

ABSTRACT

Deductive semantics is a novel software semantic theory that deduces the semantics of a program in a given programming language from a unique abstract semantic function to the concrete semantics embodied by the changes of status of a finite set of variables constituting the semantic environment of the program. There is a lack of a generic semantic function and its unified mathematical model in conventional semantics, which may be used to explain a comprehensive set of programming statements and computing behaviors. This article presents a complete paradigm of formal semantics that explains how deductive semantics is applied to specify the semantics of real-time process algebra (RTPA) and how RTPA challenges conventional formal semantic theories. Deductive semantics can be applied to define abstract and concrete semantics of programming languages, formal notation systems, and large-scale software systems, to facilitate software comprehension and recognition, to support tool development, to enable semantics-based software testing and verification, and to explore the semantic complexity of software systems. Deductive semantics may greatly simplify the description and analysis of the semantics of complicated software systems specified in formal notations and implemented in programming languages.

Keywords: computational linguistics; deductive semantics; denotational mathematics; formal semantics; mathematical model of semantics; RTPA; semantic analyses; semantic diagrams; semantic functions; semantic environment; semantics of RTPA; software engineering

INTRODUCTION

Semantics in linguistics is a domain that studies the interpretation of words and sentences, and analysis of their meanings. Semantics deals with how the meaning of a sentence in a language is obtained, hence the sentence is comprehended. Studies on semantics explore mechanisms in the understanding of languages and their meanings on the basis of syntactic structures (Chomsky, 1956, 1957, 1959, 1962, 1965, 1982; Tarski, 1944).

Software semantics in computing and computational linguistics have been recognized as one of the key areas in the development of fundamental theories for computer science and software engineering (Bjoner, 2000; Gries, 1981; Hoare, 1969; McDermid, 1991; Sloneg & Kurts, 1995; Wang, 2006b, 2007c). The semantics of a programming language is the behavioral meaning that constitute what a syntactically correct instructional statement in the language is supposed to do during run time. The development of formal semantic theories of programming is one of the pinnacles of computing and software engineering (Gunter, 1992; Meyer, 1990; Louden, 1993; Bjoner, 2000; Pagan, 1981).
Definition 1. 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 underpinning computing environment.

A number of formal semantics, such as the operational (Marcotty & Ledgard, 1986; Ollongren, 1974; Wegner, 1972; Wikstrom, 1987), denotational (Bjorner and Jones, 1982; Jones, 1980; Schmidt, 1988, 1994, 1996; Scott, 1982; Scott & Strachey, 1971), axiomatic (Dijkstra, 1975, 1976; Gries, 1981; Hoare, 1969), and algebraic (Goguen, Thatcher, Wagner, & Wright, 1977; Goguen & Malcolm, 1996; Guttag & Horning, 1978), have been proposed in the last three decades for defining and interpreting the meanings of programs and programming languages. The classic software semantics are oriented on a certain set of software behaviors that are limited at the level of language statements rather than that of programs and software systems. There is a lack of a generic semantic function and its unified mathematical model in conventional semantics, which may be used to explain a comprehensive set of programming statements and computing behaviors. 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 systematical and machine enabled semantic analysis and code generation in software engineering, the deductive semantics is developed that provides a systematic semantic analysis methodology.

Deduction is a reasoning process that discovers new knowledge or derives a specific conclusion based on generic premises such as abstract rules or principles (Wang, 2006b, 2007a, 2007c). 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 presented.

Definition 2. Deductive semantics is a formal 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.

This article presents a comprehensive theory of deductive semantics of software systems. The mathematical models of deductive semantics and the fundamental properties are described. The deductive models of semantics, semantic function, and semantic environment at various composing levels of programs are introduced. Properties of software semantics and relationships between the software behavioral space and the semantic environment are studied. New methods such as the semantic differential and semantic 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 (Wang, 2002, 2003, 2006a, 2006b, 2007a, 2007b, 2008a, 2008b) is described, where the semantics of a comprehensive set of processes is systematically modeled.

THE THEORY OF DEDUCTIVE SEMANTICS

This section presents the theory of deductive semantics (Wang, 2006b, 2007c). A generic mathematical model of deductive semantics of software is developed, and the concepts of semantic environment and semantic func-
tion are rigorously defined. Based on them, deductive semantics of programs at different composition levels are rigorously modeled. Then, common properties of software semantics are analyzed.

The Semantic Environment and Semantic Function

**Definition 3.** A semantic environment $\Theta$ of a programming language is a logical model of a set of identifiers $I$ and their values $V$ bound in pairs, i.e.:

$$\Theta \triangleq f : I \rightarrow V, V \subseteq \mathbb{R}$$

$$= \left\{ \begin{array}{l}
R(i_k, v_k) \\
\end{array} \right\}$$

$$= \left\{ (i_1, v_1), (i_2, v_2), \ldots, (i_{\#I}, v_{\#I}) \right\}$$

(1)

where $\mathbb{R}$ is the set of real numbers, $i_k \in I$, $v_k \in V \subseteq \mathbb{R}$, and $\#I$ the number of elements in $I$.

Note the big-R notation is adopted to denote a set of recurring structures or repetitive behaviors (Wang, 2002, 2007c, 2008a). The semantic environment constituting the behaviors of software is inherently a three dimensional structure known as those of operations, memory space, and time.

**Definition 4.** 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.:

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

(2)

where $OP$ is a finite set of operations, $T$ is a finite set of discrete time points of program execution, and $S$ is a finite set of memory locations or their logical representations by identifiers of variables.

According to Definitions 3 and 4, the set of variables of a program, $S$, plays an important role in semantic modeling and analysis, because they are the objects of software behavioral operations and the carriers of program semantics.

Variables can be classified as free and system variables. The former are user defined and the latter are language provided. From a functional point of view, variables can be classified into object representatives, control variables, result containers, and address locaters. The life spans or scopes of variables can be categorized as persistent, global, local, and temporal. The persistent variables are those that their lifespan are longer than the program that generates them, such as data in a database or files in a distributed network.

A new calculus introduced in deductive semantics is the partial differential of sets (Wang, 2006b, 2007c), which is used to facilitate the instantiation of abstract semantics by concrete ones, as described below.

**Definition 5.** Given two sets $X$ and $U$, $X \subseteq \mathcal{P}U$, a partial differential of $X$ on $U$ with elements $x, x \in X$, denoted by $\partial U/\partial x$, is an elicitation of interested elements from $U$ as specified in $X$, i.e.:

$$\frac{\partial U}{\partial x} \triangleq X \cap U, \ x \in X$$

$$= X, \ X \subseteq \mathcal{P}U$$

(3)

where $\mathcal{P}U$ denotes a power set of $U$.

The partial differential of sets can be easily extended to double, triple, or more generally, multiple partial differentials as defined below.

**Definition 6.** A multiple partial differential of $X_1, X_2, \ldots, X_n$ on $\mathcal{P}U$ with elements $x_1 \in X_1, x_2 \in X_2, \ldots, x_n \in X_n$, denoted by

$$\frac{\partial^n}{\partial x_1 \partial x_2 \ldots \partial x_n} U$$

is a Cartesian product of all partial differentials that select interested elements from $U$ as specified in $X_1, X_2, \ldots, X_n$ respectively, i.e.:
\[
\frac{\partial^n}{\partial x_1 \partial x_2 \ldots \partial x_n} U \triangleq X_1 \times X_2 \times \ldots \times X_n
\]

(4)

where \( X_1, X_2, \ldots, X_n \subseteq \mathbb{U} \) and \( \forall i \neq j, 1 \leq i, j \leq n, X_i \cap X_j = \emptyset \).

For example,

\[
\frac{\partial^2}{\partial x \partial y} U = X \times Y, \quad x \in X, y \in Y, \text{and } X \subseteq \mathbb{U}
\]

and

\[
\frac{\partial^3}{\partial x \partial y \partial z} U = X \times Y \times Z, \quad x \in X, y \in Y, z \in Z, \text{and } X \subseteq \mathbb{U}.
\]

On the basis of the definitions of software behavioral space and partial differential of sets, the semantic environment of software can be formally described.

**Definition 7.** The semantic environment \( \Theta \) of a program on a certain target machine is its run-time behavioral space \( \Omega \) projected onto the Cartesian plane determined by \( T \) and \( S \), i.e.:

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

\[
= \frac{\partial^2 \Omega}{\partial t \partial s} \quad (OP, T, S)
\]

\[
= T \times S
\]

(5)

As indicated in Definition 7, the semantic environment of a program is a dynamic space over time, because following each execution of a statement in the program, the semantic environment \( \Theta \), particularly the sets of variables \( S \) and their values \( V \), may be changed.

In semantic analysis, the changed part of the semantic environment \( \Theta \) is particularly interested, which is the embodiment of software semantics. 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 by mathematical deduction.

**Definition 8.** A semantic function of a program \( \varphi \), \( f_\varphi(\varphi) \), is a function that maps the semantic environment \( \Theta \) into a finite set of values \( V \) determining by a Cartesian product on a finite set of executing steps \( T \) and a finite set of variables \( S \), i.e.:

\[
f_\varphi(\varphi) \triangleq f : T \times S \rightarrow V = \left\{ \begin{array}{c}
\begin{array}{cccc}
s_1 & s_2 & \cdots & s_m \\
t_0 & \bot & \cdots & \bot \\
t_1 & v_{11} & v_{12} & \cdots & v_{1m} \\
\vdots & \vdots & \ddots & \vdots \\
t_n & v_{n1} & v_{n2} & \cdots & v_{nm}
\end{array}
\end{array} \right.
\]

(6)

where \( T = \{t_0, t_1, \ldots, t_p\} \), \( S = \{s_1, s_2, \ldots, s_m\} \), and \( V \) is a finite set of values \( v(t_i, s_j), 0 \leq i \leq n, \text{and } 1 \leq j \leq m \).

In Equation 6, all values of \( v(t_i, s_j) \) at \( t_0 \) is 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 \( v(t_i, s_j) \neq \bot \) dependent on the context of previous statement(s) or the initialization of the system.

According to Definitions 7 and 8, the semantic environment and the domain of a semantic function can be illustrated by a semantic diagram as described below (Wang, 2006b, 2007c).

**Definition 9.** A semantic diagram is a sub-Cartesian-plane in the semantic environment \( \Theta \) that forms the domain of the semantic function for a given process \( P \) with \( f_\varphi(P) = f : T \times S_p \rightarrow V_p \).

For example, the semantic diagram of an abstract process \( P, f_\varphi(P) \), as defined in Definition 9 can be illustrated in Figure 1, where \( V_p \) is the domain of dynamic variable values of process \( P \) over time, i.e., \( V_p = T_p \times S_p \). The semantic

---

Copyright © 2008, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
The semantic diagram can be used to analyze complex semantic relations, and to demonstrate semantic functions and their semantic environments. Observing Figures 1 and 3, the flowing properties of process relations can be derived.

Lemma 1. The variables of two arbitrary processes P and Q, S_P and S_Q, in the semantic environment Θ 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:
   \[ S_L = S - S_G \subseteq S_P \oplus S_Q, \]

where \( S_{tp} = S_L \setminus S_Q \) and \( S_{tq} = S_L \setminus S_P \).

Deductive Semantics of Programs at Different Levels of Compositions

According to the generic model and the hierarchical architecture of programs, the semantics of a program in a given programming language can be described and analyzed at various composition levels, such as those of statement, process, and system from the bottom-up (Wang, 2007c, 2008b).

Definition 10. The semantics of a statement \( p, \theta(p) \), on a given semantic environment Θ is a double partial differential of the semantic function \( f_\theta(p) \) on executing steps T and the set of variables S, i.e.:

\[
\theta(P) \triangleq \frac{\partial^2 f_\theta(p)}{\partial t \partial s} \frac{#T(p)}{\#T(p)} \frac{#S}{#S(p)} \\
= R \begin{cases} 
R \begin{pmatrix} v_p t_1, s_j \end{pmatrix} \\
(=0) \quad (j=1, \ldots, m) 
\end{cases} \\
= \left\{ \begin{array}{c}
\begin{bmatrix}
t_0 & v_{01} & v_{02} & \cdots & v_{0m} \\
t_{t_1} & v_{11} & v_{12} & \cdots & v_{1m}
\end{bmatrix}
\end{array} \right\}
\]  

where \( t \) denotes the discrete time before and after the execution of \( p \) during \( (t_p, t) \), 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 10, the first partial differential selects all related variable \( S(p) \) of the statement \( p \) from \( \Theta \). The second partial differential selects a set of discrete steps of \( p \)’s execution \( T(p) \) from \( \Theta \). According to Definition 10, the semantics of a statement can be reduced onto a semantic function that results in a 2-D matrix with the changes of values of all variables over time of program execution.

On the basis of Definitions 8 and 10, semantics of individual statements can be analyzed using Equation 10 in a deductive process.

Example 1. Analyze the semantics of Statement 3, \( \theta(p), \) in the following program entitled sum::

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

According to Definition 10, the semantics of Statement \( p \) is as follows:

\[
\theta(p) = \frac{\partial^2}{\partial t \partial s} f_0(p) \\
= R \left( R v_{p_i} t_{i,j} \right) \\
= R \left( R v_{p_i} t_{i,j} \right) \\
= \left( \begin{array}{cccc}
  x & y & z \\
  t_2 & 8 & 2 & \perp \\
  (t_2, t_3) & 8 & 2 & 10
\end{array} \right)
\]

This example shows how the concrete semantics of a statement can be derived on the basis of the generic and abstract semantic function as given in Definition 10.

**Definition 11.** The semantic effect of a statement \( p \), \( \theta^*(p) \), is the resulted changes of values of variables by its semantic function \( \theta(p) \) during the time interval immediately before and after the execution of \( p \), \( \Delta t = (t_i, t_{i+1}) \), see Box 1, where \( \rightarrow \) denotes a transition of values for a given variable.

**Example 2.** For the same statement \( p \) as given in Example 1, determine its semantic effect \( \theta^*(p) \).

According to Equation 12, the semantic effect \( \theta^*(p_j) \) is seen in Box 2.

It is noteworthy in Examples 1 and 2 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.

Considering that a program or a process is composed by individual statements with given rules of compositions, the definition and mathematical model of deductive semantics at the statement level can be extended onto the higher levels of program hierarchy.

**Box 1.**

\[
\theta^*(p) = \frac{\#S(p)}{R} \left( v_p(t_{i,j}) \oplus v_p(t_{i+1,j}) \right) \\
= R \left( v_p(t_{i,j}) \rightarrow v_p(t_{i+1,j}) \right) \left( v_p(t_{i,j}) \neq v_p(t_{i+1,j}) \right)
\]

**Box 2.**

\[
\theta^*(p_j) = R \left( v_{p_j}(t_{2,j}) \rightarrow v_{p_j}(t_{3,j}) \right) \left( v_{p_j}(t_{2,j}) \neq v_{p_j}(t_{3,j}) \right) \\
= \left( < v_{p_j}(t_2, z) \rightarrow v_{p_j}(t_3, z) \right) \left( < v_{p_j}(t_2, z) \neq v_{p_j}(t_3, z) \right) \\
= \{ < v_{p_j}(t_2, z) \rightarrow v_{p_j}(t_3, z) = 10 > \}
\]
**Definition 12.** 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 \( T \); see Box 3, 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 12, the semantics of a program at the top-level composition can be deduced to the combination of the semantics of a set of processes, each of which can be further deduced to the composition of all statements’ semantics as described below.

**Definition 13.** The semantics of a program \( \Phi, \Theta(\Phi) \), 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.:

\[
\Theta(\Phi) = \sum_{k=1}^{\#K(\Phi)} \frac{\partial^2}{\partial t \partial s} f_{\theta}(\Phi) \\
= \sum_{k=1}^{\#K(\Phi)} \sum_{i=0}^{\#T(P_k)} \sum_{j=1}^{\#S(P_k)} R_{i, j, k} \cdot v_{p_k}(t_i, s_j)
\]

(14)

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

It is noteworthy that Equation 14 will usually result in a very large matrix of semantic space, which can be quantitatively predicated as follows.

**Definition 14.** The semantic space of a program \( S_{\Theta}(\Phi) \) is a product of \( \#S(\Phi) \) variables and \( \#T(\Phi) \) executing steps, i.e.:

\[
S_{\Theta}(\Phi) = \#S(\Phi) \times \#T(\Phi)
\]

\[
= \sum_{k=1}^{\#K(\Phi)} \sum_{k=1}^{\#K(\Phi)} R_{i, j, k} \cdot v_{p_k}(t_i, s_j)
\]

(15)

The semantic space of programs provides a useful measure of 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.

**Properties of Software Semantics**

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

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

**Theorem 1.** 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.

Theorem 1 asserts that, for most non real-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 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.

**Definition 15.** The behavior of a computational statement is a set of observable actions or changes of status of objects operated by the statement.

According to Definition 4, the behavioral space of software, $\Omega$, is three dimensional, while as given in Definition 7, 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 Figure 2.

The theory of deductive semantics can be systematically applied to formally and rigorously model and describe the semantics of the RTPA metaprocesses and the process relations (operations). On the basis of the mathematical models and properties of deductive semantics, the following sections formally describe a comprehensive set of RTPA semantics, particularly the 17 metaprocesses and the 17 process relations (Wang, 2002, 2003, 2007c, 2008a, 2008b). This work extends the coverage of semantic rules of programming languages to a complete set of features that encompasses both basic computing operations and their algebraic composition rules. Because RTPA is a denotational mathematical structure based on process algebra that covers a comprehensive set of computing and programming requirements, any formal semantics that is capable to process RTPA is powerful enough to express the semantics of any programming language.

**DEDUCTIVE SEMANTICS OF RTPA METAPROCESSES**

Metaprocesses of RTPA are elicited from basic computational requirements. Complex

---

*Figure 2. Relationship between software behavior space and the semantic environment*
processes can be composed with multiple metaprocesses. RTPA identified 17 metaprocesses, \( \mathcal{P} \), on fundamental computing operations such as assignment, system control, event/time handling, memory and I/O manipulation, i.e., \( \mathcal{P} = \{ \text{:=, } \implies, \iff, \leq, \geq, <, >, \langle, \rangle, \leftarrow, \rightarrow, \uparrow, \downarrow, !, \odot, \bigotimes, \|$ \}. Detailed descriptions of the metaprocesses of RTPA and their syntaxes may be referred to (Wang, 2002, 2007c, 2008b), where each metaprocess is a basic operation on one or more operands such as variables, memory elements, or I/O ports. Based on Definitions 8 and 12, the deductive semantics of the set of RTPA metaprocesses can be defined in the following subsections.

The Assignment Process

Definition 16. The semantics of the assignment process on a given semantic environment \( \Theta \), \( \Theta[\mathbf{RT} := x\mathbf{RT}] \), is a double partial differential of the semantic function \( f_\mathbf{y}(\mathbf{RT} := x\mathbf{RT}) \) on the sets of variables \( S \) and executing steps \( T \), i.e.:

\[
\theta[\mathbf{RT} := x\mathbf{RT}] = \frac{\partial^2}{\partial t \partial s} f_\mathbf{y}(\mathbf{RT} := x\mathbf{RT})
\]

\[
\theta[\mathbf{RT} := x\mathbf{RT}] = \begin{cases} 
R_{j=0}^{1} & v(t_j, s_j) \\
R_{j=1}^{2} & (t_0, t_1) 
\end{cases}
\]

\[
= \begin{pmatrix} 
x\mathbf{RT} & y\mathbf{RT} 
x\mathbf{RT} & y\mathbf{RT} 
\end{pmatrix}
\]

(16)

where the size of the matrix is \( \#T \cdot \#S \).

The Evaluation Process

Definition 17. The semantics of the evaluation process on \( \Theta \), \( \Theta[\text{exp}\mathbb{T} \rightarrow \mathbb{T}] \), is a double partial differential of the semantic function \( f_\text{exp}(\text{exp}\mathbb{T} \rightarrow \mathbb{T}) \) on the sets of variables \( S \) and executing steps \( T \) in the following two forms, i.e.:

\[
\theta(\text{exp}\mathbb{B} \rightarrow \mathbb{B}L) = \frac{\partial^2}{\partial t \partial s} f_\text{exp}(\text{exp}\mathbb{B} \rightarrow \mathbb{B}L)
\]

\[
\theta(\text{exp}\mathbb{T} \rightarrow \mathbb{T}) = \frac{\partial^2}{\partial t \partial s} f_\text{exp}(\text{exp}\mathbb{T} \rightarrow \mathbb{T})
\]

\[
\left( \begin{array}{c}
\delta(\text{exp}\mathbb{B}\mathbb{L}) \mathbb{L} \\
\delta(\text{exp}\mathbb{T}\mathbb{T}) \mathbb{T}
\end{array} \right)
\]

(17a)

or

\[
\left( \begin{array}{c}
\delta(\text{exp}\mathbb{T}\mathbb{T}) \mathbb{T} \\
\delta(\text{exp}\mathbb{T}\mathbb{T}) \mathbb{T}
\end{array} \right)
\]

(17b)

where \( \delta(\text{exp}\mathbb{B}) \) is the Boolean evaluation function on \( \text{exp}\mathbb{B} \) that results in \( \mathbb{T} \) or \( \mathbb{F} \). \( \delta(\text{exp}\mathbb{T}) \) is a more general cardinal or numerical evaluation function on \( \text{exp}\mathbb{T} \) that results in \( \mathbb{T} \in \{ \mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{B} \} \), i.e., in types of nature number, integer, real number, and byte, respectively (Wang, 2002).

The Addressing Process

Definition 18. The semantics of the addressing process on \( \Theta \), \( \Theta[\text{id}\mathbb{S} \rightarrow \text{ptr}\mathbb{P}] \), is a double partial differential of the semantic function \( f_\text{id}(\text{id}\mathbb{S} \rightarrow \text{ptr}\mathbb{P}) \) on the sets of variables \( S \) and executing steps \( T \), i.e.:
\[ \theta (idS \Rightarrow ptrP) \triangleq \frac{\partial^2}{\partial t \partial s} f_0 (idS \Rightarrow ptrP) \]

\[ \triangledown T (idS \Rightarrow ptrP) \triangledown S (idS \Rightarrow ptrP) \]

\[ = \frac{1}{R} \frac{2}{R} v(t_i, s_j) \]

\[ = \begin{pmatrix} idS & ptrP \\ t_0 & idS \bot \hline (t_0, t_1) & idS \pi (idS)H \end{pmatrix} \]

(18)

where \( \pi (idS)H \) is a function that associates a declared identifier \( idS \) to its hexadecimal memory address located by the pointed \( ptrP \).

### The Memory Allocation Process

**Definition 19.** The semantics of the memory allocation process on \( \Theta \), \( \theta (idS \Leftarrow MEM(ptrP)RT) \), is a double partial differential of the semantic function \( f_0 (idS \Leftarrow MEM(ptrP)RT) \) on the sets of variables \( S \) and executing steps \( T \), see Box 4. Where \( \pi (idS)H \) is a mapping function that associates an identifier \( idS \) to a memory block starting at a hexadecimal address located by the pointed \( ptrP \). The ending address of the allocated memory block, \( ptrP + \text{size}(RT)-1 \), is dependent on a machine implementation of the size of a given variable in type \( RT \).

### The Memory Release Process

**Definition 20.** The semantics of the memory release process on \( \Theta \), \( \theta (idS \Leftarrow MEM(\bot)RT) \), is a double partial differential of the semantic function \( f_0 (idS \Leftarrow MEM(\bot)RT) \) on the sets of variables \( S \) and executing steps \( T \), i.e.:

\[ \theta (idS \Leftarrow MEM(\bot)RT) \triangleq \frac{\partial^2}{\partial t \partial s} f_0 (idS \Leftarrow MEM(\bot)RT) \]

\[ \triangledown T (idS \Leftarrow MEM(\bot)RT) \triangledown S (idS \Leftarrow MEM(\bot)RT) \]

\[ = \frac{1}{R} \frac{2}{R} v(t_i, s_j) \]

\[ = \begin{pmatrix} idRT & ptrP & MEMRT \\ t_0 & idS \pi (idS)H & MEM[ptrP]RT \hline (t_0, t_1) & \bot & \bot & \bot \end{pmatrix} \]

(20)

### The Read Process

**Definition 21.** The semantics of the read process on \( \Theta \), \( \theta (MEM(ptrP)RT \triangleright xRT) \), is a double partial differential of the semantic function \( f_0 (MEM(ptrP)RT \triangleright xRT) \) on the sets of variables \( S \) and executing steps \( T \), see Box 5.

Box 4.

\[ \theta (idS \Leftarrow MEM[ptrP]RT) \triangleq \frac{\partial^2}{\partial t \partial s} f_0 (idS \Leftarrow MEM[ptrP]RT) \]

\[ \triangledown T (idS \Leftarrow MEM[ptrP]RT) \triangledown S (idS \Leftarrow MEM[ptrP]RT) \]

\[ = \frac{1}{R} \frac{3}{R} v(t_i, s_j) \]

\[ = \begin{pmatrix} idS & ptrP & MEMRT \\ t_0 & idS \pi (idS)H & MEM[ptrP]RT \hline (t_0, t_1) & \bot & \bot & \bot \end{pmatrix} \]

(19)
The Write Process

Definition 22. The semantics of the write process on $\Theta$, $\Theta(\text{MEM}(\text{ptr}P|RT \leftarrow xRT))$, is a double partial differential of the semantic function $f_\theta(\text{MEM}(\text{ptr}P|RT \leftarrow xRT))$ on the sets of variables $S$ and executing steps $T$, i.e.: 

$$\theta(\text{MEM}(\text{ptr}P|RT \leftarrow xRT)) = \frac{\partial^2}{\partial t \partial s} f_\theta(\text{MEM}(\text{ptr}P|RT \leftarrow xRT)) \mid_{s \neq 0} \neq S(\text{MEM}(\text{ptr}P|RT \leftarrow xRT))$$

$$= \left\{ \begin{array}{ll} t_0 & xRT \\
(t_0, t_1) & xRT \end{array} \right. \quad \text{MEM}(\text{ptr}P|RT) \quad xRT$$

(22)

The Input Process

Definition 23. The semantics of the input process on $\Theta$, $\Theta(\text{PORT}(\text{ptr}P|RT \triangleright xRT))$, is a double partial differential of the semantic function $f_\theta(\text{PORT}(\text{ptr}P|RT \triangleright xRT))$ on the sets of variables $S$ and executing steps $T$, i.e.: 

$$\theta(\text{PORT}(\text{ptr}P|RT \triangleright xRT)) = \frac{\partial^2}{\partial t \partial s} f_\theta(\text{PORT}(\text{ptr}P|RT \triangleright xRT)) \mid_{s \neq 0} \neq S(\text{PORT}(\text{ptr}P|RT \triangleright xRT))$$

$$= \left\{ \begin{array}{ll} \text{ptrP} & \text{PORT}(\text{ptr}P|RT) \\
(t_0, t_1) & \text{PORT}(\text{ptr}P|RT) \quad \text{PORT}(\text{ptr}P|RT) \
\end{array} \right. \quad xRT$$

(23)

The Output Process

Definition 24. The semantics of the output process on $\Theta$, $\Theta(xRT | \triangleright PORT(\text{ptr}P|RT))$, is a double partial differential of the semantic function $f_\theta(xRT | \triangleright PORT(\text{ptr}P|RT))$ on the sets of variables $S$ and executing steps $T$, i.e.: 

$$\theta(xRT | \triangleright PORT(\text{ptr}P|RT)) = \frac{\partial^2}{\partial t \partial s} f_\theta(xRT | \triangleright PORT(\text{ptr}P|RT)) \mid_{s \neq 0} \neq S(xRT | \triangleright PORT(\text{ptr}P|RT))$$

$$= \left\{ \begin{array}{ll} \text{ptrP} & \text{MEM}(\text{ptr}P|RT) \\
(t_0, t_1) & \text{MEM}(\text{ptr}P|RT) \quad \text{MEM}(\text{ptr}P|RT) \
\end{array} \right. \quad xRT$$

(21)
\[ \theta(\text{PORT}[\text{ptrP}]) \]
\[ \quad \triangleq \frac{\partial^2}{\partial t \, \partial s} f_0(\text{PORT}[\text{ptrP}]) \]
\[ \quad = \begin{cases} R_{i=0} & (R_{j=1} v(t_i, s_j)) \\ \frac{1}{2} R_{i=0} R_{j=1} v(t_i, s_j) & \end{cases} \]
\[ \text{The Timing Process} \]

**Definition 25.** The semantics of the timing process on \( \Theta \), \( \theta(\triangledown TM \triangledown \triangledown TM) \), is a double partial differential of the semantic function \( f_0(\triangledown TM \triangledown \triangledown TM) \) on the sets of variables \( S \) and executing steps \( T \), i.e.:
\[ \theta(\triangledown TM \triangledown \triangledown TM) \triangleq \frac{\partial^2}{\partial t \, \partial s} f_0(\triangledown TM \triangledown \triangledown TM) \]
\[ = \begin{cases} R_{i=0} & (R_{j=1} v(t_i, s_j)) \\ \frac{1}{2} R_{i=0} R_{j=1} v(t_i, s_j) & \end{cases} \]
\[ \text{The Increase Process} \]

**Definition 27.** The semantics of the increase process on \( \Theta \), \( \theta(\uparrow (\text{PORT})) \), is a double partial differential of the semantic function \( f_0(\uparrow (\text{PORT})) \) on the sets of variables \( S \) and executing steps \( T \), i.e.:
\[ \theta(\uparrow (\text{PORT})) \triangleq \frac{\partial^2}{\partial t \, \partial s} f_0(\uparrow (\text{PORT})) \]
\[ = \begin{cases} R_{i=0} & (R_{j=1} v(t_i, s_j)) \\ \frac{1}{2} R_{i=0} R_{j=1} v(t_i, s_j) & \end{cases} \]

**The Duration Process**

**Definition 26.** The semantics of the duration process on \( \Theta \), \( \theta(\triangledown TM \triangledown \triangledown TM + \Delta z) \), is a double partial differential of the semantic function \( f_0(\triangledown TM \triangledown \triangledown TM + \Delta z) \) on the sets of variables \( S \) and executing steps \( T \), i.e.:
\[ \theta(\triangledown TM \triangledown \triangledown TM + \Delta z) \triangleq \frac{\partial^2}{\partial t \, \partial s} f_0(\triangledown TM \triangledown \triangledown TM + \Delta z) \]
\[ = \begin{cases} R_{i=0} & (R_{j=1} v(t_i, s_j)) \\ \frac{1}{2} R_{i=0} R_{j=1} v(t_i, s_j) & \end{cases} \]
\[ \text{The Decrease Process} \]

**Definition 28.** The semantics of the decrease process on \( \Theta \), \( \theta(\downarrow (\text{PORT})) \), is a double partial differential of the semantic function \( f_0(\downarrow (\text{PORT})) \) on the sets of variables \( S \) and executing steps \( T \), i.e.:
\[ \theta(\downarrow (\text{PORT})) \triangleq \frac{\partial^2}{\partial t \, \partial s} f_0(\downarrow (\text{PORT})) \]
\[ = \begin{cases} R_{i=0} & (R_{j=1} v(t_i, s_j)) \\ \frac{1}{2} R_{i=0} R_{j=1} v(t_i, s_j) & \end{cases} \]
The Skip Process

**Definition 30.** The semantics of the skip process on \( \Theta, \theta(\otimes) \), is a double partial differential of the semantic function \( f_\theta(\otimes) \) on the sets of variables \( S \) and executing steps \( T \), i.e.:

\[
\theta(\otimes) \triangleq \theta(P^k \cap P^{k-1})
\]

\[
= \frac{\partial^2}{\partial t \partial s} f_\theta(P^k \cap P^{k-1})
\]

\[
\overset{T(P^k \cap P^{k-1}) \overset{S(P^k \cap P^{k-1})}{\quad}}{R \quad R \quad R \quad v(t_i, s_j)}
\]

\[
= R \quad (R \quad v(t_i, s_j))
\]

\[
= \frac{1}{2} \quad R \quad R \quad v(t_i, s_j)
\]

\[
= \left\{ \begin{array}{l}
S_{P^{k-1}} \quad S_{P^k} \\
S_{P^{k-1}} \quad S_{P^k}
\end{array} \right.
\]

\[
\theta(\otimes) \triangleq \theta(P^k \cap P^{k-1})
\]

(31)

where \( P^k \) is a process \( P \) at a given embedded layer \( k \) in a program where \( P^0 \) at the uttermost layer, and \( \cap \) denotes the jump process relation where its semantics will be formally defined in the next section.

According to Definition 30, the skip process \( \otimes \) has no semantic effect on the current process \( P^k \) at the given embedded layer \( k \) in a program, such as a branch, loop, or function. However, it redirects the system to jump to execute an upper-layer process \( P^{k-1} \) in the embedded hierarchy. Therefore, skip is also known as *exit* or *break* in programming languages.

The Stop Process

**Definition 31.** The semantics of the stop process on \( \Theta, \theta(\uparrow) \), is a double partial differential of the semantic function \( f_\theta(\uparrow) \) on the sets of variables \( S \) and executing steps \( T \), i.e.:

\[
\theta(\uparrow) \triangleq \theta(@eS) \iff \text{PORT}[ptrP]S
\]

(30)
DEDUCTIVE SEMANTICS OF RTPA PROCESS RELATIONS

The preceding section provides formal definitions of metaprocesses of RTPA for software system modeling. Via the composition of multiple metaprocesses by the 17 process relations, complex architectures and behaviors of software systems, in the most complicated case, a real-time system, can be sufficiently described (Wang, 2002, 2006a, 2007c, 2008b). On the basis of Definitions 8 and 12, the semantics of the RTPA process relations can be formally defined and analyzed as follows.

The Sequential Process Relation

Definition 32. The semantics of the sequential relation of processes on $\Theta$, $\theta(P \rightarrow Q)$, is a double partial differential of the semantic function $f_\theta(P \rightarrow Q)$ on the sets of variables $S$ and executing steps $T$, i.e.:

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

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

where the stop process $\Box$ does nothing but returns the control of execution to the system.

Example 3. Analyze the semantics of the sequential processes $P_0$ through $P_5$ in the following program:

```c
void sequential_sum;
{
    int x, y, z;
    // P_0
```
\[x = 2; \quad y = 8; \quad z := x + y; \quad z := x + y + z; \quad \text{print } z;\]

According to Definition 32, the semantics of the above program can be analyzed as seen in Box 6. Where \(\text{PORT}[\text{CRTP}]\mathbb{N}\) denotes a system monitor of type \(\mathbb{N}\) located by the pointer \(\text{CRTP}\).

### The Jump Process Relation

**Definition 33.** The semantics of the jump relations of processes on \(\Theta\), \(\theta(P \bowtie Q)\), is a double partial differential of the semantic function \(f_\theta(P \bowtie Q)\) on the sets of variables \(S\) and executing steps \(T\), i.e.:

\[
\theta(P_0 \rightarrow P_1 \rightarrow \ldots \rightarrow P_4) = \frac{\partial^2}{\partial t \partial s} f_\theta(P_0 \rightarrow P_1 \rightarrow \ldots \rightarrow P_4)
\]

\[
= \frac{\partial^2}{\partial t \partial s} f_\theta(P_0) \rightarrow \frac{\partial^2}{\partial t \partial s} f_\theta(P_1) \rightarrow \ldots \rightarrow \frac{\partial^2}{\partial t \partial s} f_\theta(P_4)
\]

\[
= R_{j=0}^\theta R_{j=1}^\theta R_{j=2}^\theta \ldots R_{j=4}^\theta v_{t_j, s_j}
\]

\[
= \begin{array}{ccccccc}
  x & y & z & \text{PORT}[\text{CRTP}]\mathbb{N} \\
  t_0 & \bot & \bot & \bot & \bot \\
  (t_0, t_1) & 2 & \bot & \bot & \bot \\
  (t_1, t_2) & 2 & 8 & \bot & \bot \\
  (t_2, t_3) & 2 & 8 & 10 & \bot \\
  (t_3, t_4) & 2 & 8 & 20 & \bot \\
  (t_4, t_5) & 2 & 8 & 20 & 20 \\
\end{array}
\]

(34)
\[
0(P \cap Q) = \frac{\partial^2}{\partial t \partial s} f_0(P \cap Q)
\]
\[
= \frac{\partial^2}{\partial t \partial s} f_0(P) \left( \cap \frac{\partial^2}{\partial t \partial s} f_0(Q)
\right)
\]
\[
= R \left( R_{i,j} v_{i,j} t_s \right) \cap R \left( R_{i,j} v_{i,j} t_s \right)
\]
\[
= R \left( R_{i,j} v_{i,j} t_s \right)
\]
\[
= \left\{ \begin{array}{c}
S_P \quad S_Q \quad S_{PQ} \quad addr_H \\
[t_0, t_1] \quad V_{iP} \quad \perp \quad V_{iQ} \quad \perp \\
(t_1, t_2) \quad \perp \quad \quad \pi(S_Q)H \\
(t_2, t_3) \quad V_{iQ} \quad V_{iQ}
\end{array} \right\}
\]
(35)

where \(\pi(S_Q)H\) is a system addressing function of the system that directs the program control flow to execute the new process \(Q\), which physically located in a different memory address at \(addr_H = \pi(S_Q)H\).

The semantic diagram of the jump process relation as defined in Equation 35 is illustrated in Figure 4 on \(\Theta(P \cap Q)\).

The jump process relation is an important process relation that forms a fundamental part of many other processes and constructs. For instances, the jump process relation has been applied in expressing the semantics of the skip and stop processes in the preceding section.

The Branch Process Relation

**Definition 34.** The semantics of the branch relation of processes on \(\Theta, \Theta(\text{exp BL} = T \rightarrow P \ | \ \text{exp BL} \rightarrow Q)\), abbreviated by \(\Theta(P \mid Q)\), is a double partial differential of the semantic function \(f_0(P \mid Q)\) on the sets of variables \(S\) and executing steps \(T\), see Box 7, where \(\delta(\text{exp BL})\) is the evaluation function on the value of \(\text{exp BL}\), \(\delta(\text{exp BL}) \in \{T, F\}\).

The semantic diagram of the branch process relation as defined in Equation 34 is illustrated in Figure 5 on \(\Theta(\text{exp BL} \rightarrow P \mid \text{exp BL} \rightarrow Q)\).
where $V_G$ is a set of global variables shared by $P_0$, $P_P$, and $P_{n-1}$.

The semantic diagram of the switch process relation as defined in Equation 37 is illustrated in Figure 6 on $\Theta(\text{exp}^\text{BT} \rightarrow P | \text{~} \rightarrow \emptyset)$.

**The While-Loop Process Relation**

**Definition 36.** The semantics of the while-loop relations of processes on $\Theta$,

$$\theta(\text{exp}^\text{BT} \rightarrow P | \text{~} \rightarrow \emptyset) = \frac{\partial^2}{\partial t \partial s} f_\theta(\text{exp}^\text{BT} \rightarrow P | \text{~} \rightarrow \emptyset)$$

is a double partial differential of the semantic function

$$f_\theta(\text{exp}^\text{BT} \rightarrow P | \text{~} \rightarrow \emptyset)$$

on the sets of variables $S$ and executing steps $T$, i.e.:
\[ \theta(\frac{d^2}{dt \, ds} \frac{\partial}{\partial s} f_\theta(\mathbf{R}^n(P))) = \frac{\partial}{\partial \mathbf{R}^n(P)} \mathbf{R}(P) \]
\[ = \mathbf{R}(\frac{\partial^2}{\partial t \, \partial s} f_\theta(P)) \]
\[ = \mathbf{R}(\frac{\partial^2}{\partial t \, \partial s} f_\theta(P)) \]
\[ = \mathbf{R}(\frac{\partial^2}{\partial t \, \partial s} f_\theta(P)) \]
\[ = \mathbf{R}(\frac{\partial^2}{\partial t \, \partial s} f_\theta(P)) \]
\[ \begin{align*}
\exp^{\text{BL}} & S_P \\
[t_0, t_1] & \delta(\exp^{\text{BL}}) \perp \\
(t_1, t_2] & \mathbf{T} \quad V_p \\
(t_1, t_2] & \mathbf{F} \otimes \\
\vdots \quad \vdots \quad \vdots \\
(t_1, t_4] & \delta(\exp^{\text{BL}}) \ominus \\
(t_4, t_5] & \mathbf{T} \quad V_p \\
(t_4, t_5] & \mathbf{F} \otimes \\
\end{align*} \]
\[ (38) \]

where \( \perp \) denotes exit, and \( \delta(\exp^{\text{BL}}) \) is the evaluation function on the Boolean expression, \( \delta(\exp^{\text{BL}}) \in \{ \mathbf{T}, \mathbf{F} \} \).

The semantic diagram of the while-loop process relation as defined in Equation 38 is illustrated in Figure 7 on \( \Theta \).

**The Repeat-Loop Process Relation**

**Definition 37.** The semantics of the repeat-loop relations of processes on \( \Theta \),

\[ \theta(\frac{d^2}{dt \, ds} \frac{\partial}{\partial s} f_\theta(\mathbf{R}^n(P))) = \frac{\partial}{\partial \mathbf{R}^n(P)} \mathbf{R}(P) \]

is a double partial differential of the semantic function

\[ f_\theta(\mathbf{R}^n(P)) \]

on the sets of variables \( S \) and executing steps \( T \), i.e.:

\[ \begin{align*}
\exp^{\text{BL}} & S_P \\
[t_0, t_1] & \perp \quad V_p \\
(t_1, t_2] & \delta(\exp^{\text{BL}}) \ominus \\
(t_2, t_3] & \mathbf{T} \quad V_p \\
(t_2, t_3] & \mathbf{F} \otimes \\
\vdots \quad \vdots \quad \vdots \\
(t_4, t_5] & \delta(\exp^{\text{BL}}) \ominus \\
(t_5, t_6] & \mathbf{T} \quad V_p \\
(t_5, t_6] & \mathbf{F} \otimes \\
\end{align*} \]
\[ (39) \]

The semantic diagram of the repeat-loop process relation as defined in Equation 39 is illustrated in Figure 8 on \( \Theta \).

**The For-Loop Process Relation**

**Definition 38.** The semantics of the for-loop relations of processes on \( \Theta \),

\[ \theta(\mathbf{R}^n(P(i))) \]
is a double partial differential of the semantic function
\[ f_\theta(P \mapsto Q) \]
on the sets of variables S and executing steps T, i.e.:
\[ \Theta(P \mapsto Q) \triangleq \frac{\partial^2}{\partial t \partial s} f_\theta(P \mapsto Q) \]
\[ = \frac{\partial^2}{\partial t \partial s} f_\theta(P) \mapsto Q \]
\[ = R \left( R^n_{i=0} R_{j=1}^{s(T(P) \neq S(P))} v_{P_i}(t_i, s_j) \right) \]
\[ = \left\{ \begin{array}{c}
    \{t_0, t_1, \ldots, t_n\} \quad \perp \quad 1 \\
    \{t_1, t_2, \ldots, t_n\} \quad V_{P_1} \\
    \vdots \quad \vdots \\
    \{t_{n-2}, t_{n-1}\} \quad n \\
    \{t_{n-1}, t_n\} \quad n \quad V_{P_n}
\end{array} \right. \]  
(40)

The semantic diagram of the procedure call process relation as defined in Equation 41 is illustrated in Figure 10 on \( \Theta(P \mapsto Q) \).

**The Recursive Process Relation**

**Definition 40.** The semantics of the recursive relations of processes on \( \Theta, \Theta(P \mapsto P) \), is a double partial differential of the semantic function
\[ f_\theta(P \mapsto P) \]
on the sets of variables S and executing steps T, i.e.:
\[ \Theta(P \mapsto P) \triangleq \frac{\partial^2}{\partial t \partial s} f_\theta(P \mapsto P) \]
\[ = \frac{\partial^2}{\partial t \partial s} f_\theta(P) \mapsto P \]
\[ = R \left( R^n_{i=0} R_{j=1}^{s(T(P) \neq S(P))} v_{P_i}(t_i, s_j) \right) \]
\[ = \left\{ \begin{array}{c}
    \{t_0, t_1, \ldots, t_n\} \quad \perp \quad 1 \\
    \{t_1, t_2, \ldots, t_n\} \quad V_{P_1} \\
    \vdots \quad \vdots \\
    \{t_{n-2}, t_{n-1}\} \quad n \\
    \{t_{n-1}, t_n\} \quad n \quad V_{P_n}
\end{array} \right. \]  
(41)

The semantic diagram of the for-loop process relation as defined in Equation 40 is illustrated in Figure 9 on \( \Theta \).

**The Function Call Process Relation**

**Definition 39.** The semantics of the function call relations of processes on \( \Theta, \Theta(P \mapsto Q) \),

\[ \Theta(P \mapsto Q) \triangleq \frac{\partial^2}{\partial t \partial s} f_\theta(P \mapsto Q) \]
\[ = \frac{\partial^2}{\partial t \partial s} f_\theta(P) \mapsto Q \]
\[ = R \left( R^n_{i=0} R_{j=1}^{s(T(Q) \neq S(Q))} v_{P_i}(t_i, s_j) \right) \]
\[ = \left\{ \begin{array}{c}
    \{t_0, t_1, \ldots, t_n\} \quad \perp \quad 1 \\
    \{t_1, t_2, \ldots, t_n\} \quad V_{P_1} \\
    \vdots \quad \vdots \\
    \{t_{n-2}, t_{n-1}\} \quad n \\
    \{t_{n-1}, t_n\} \quad n \quad V_{P_n}
\end{array} \right. \]  
(40)
The semantic diagram of the recursive process relation as defined in Equation 42 is illustrated in Figure 11 on $\Theta(P \circ P)$.

\[ \Theta(P \circ P) \overset{\Delta}{=} \frac{\partial^2}{\partial t \partial s} f_\theta(P \circ P) \]
\[ = \frac{\partial^2}{\partial t \partial s} f_\theta(P) \cup \frac{\partial^2}{\partial t \partial s} f_\theta(Q) \]
\[ = R_{i=0}^T \left( R_{j=1}^T v_t, s_j \right) \cup R_{i=0}^T \left( R_{j=1}^T v_t, s_j \right) \]
\[ = R_{i=0}^T \left( R_{j=1}^T v_t, s_j \right) \]
\[ = \left\{ \begin{array}{c}
\left[ t_0, t_1 \right] \ V_{p_0} \\
\left[ t_1, t_2 \right] \ V_{p_1} \\
\vdots \\
\left[ t_{k-1}, t_k \right] \ V_{p_{k-1}} \\
\left[ t_k, t_{k+1} \right] \ V_{p_{k+1}} \\
\end{array} \right \} \\
\left( t_0, t_1 \right) \ V_{p_0} S_p \quad \left( t_0, t_1 \right) \ V_{p_0} S_Q \quad \left( t_0, t_1 \right) \ V_{p_0} S_{PQ} \]
\[ = \frac{\partial^2}{\partial t \partial s} f_\theta(P) \bigg|_{s \leq \#T(P)} \cup \frac{\partial^2}{\partial t \partial s} f_\theta(Q) \bigg|_{s \leq \#T(Q)} \]
\[ = R_{i=0}^T \left( R_{j=1}^T v_t, s_j \right) \bigg|_{s \leq \#T(P) \#T(Q)} \bigg|_{s \leq \#S(P \cup Q)} \]
\[ = \left\{ \begin{array}{c}
\left[ t_0, t_1 \right] \ V_{p_0} \quad V_{Q_0} \quad V_{Q_0} \quad V_{Q_0} \\
\vdots \\
\left[ t_{k-1}, t_k \right] \ V_{p_{k-1}} \quad V_{Q_{k-1}} \quad V_{Q_{k-1}} \quad V_{Q_{k-1}} \\
\end{array} \right \} \\
\right. \]
\[ \text{(42)} \]

where $t_s = \max(\#T(P)), (\#T(Q))$ is the synchronization point between two parallel processes.

The semantic diagram of the parallel process relation as defined in Equation 43 is illustrated in Figure 12 on $\Theta(P || Q)$.

It is noteworthy that parallel processes $P$ and $Q$ are interlocked. That is, they should start and end at the same time. In case $t_1 \neq t_2$, the process completed earlier, should wait for the completion of the other. The second condition between parallel processes is that the shared resources, in particular variables, memory space, ports, and devices should be protected. That is, when a process operates on a shared resource, it is locked to the other process until

\[ \Theta(P || Q) \overset{\Delta}{=} \frac{\partial^2}{\partial t \partial s} f_\theta(P || Q) \]
\[ = \frac{\partial^2}{\partial t \partial s} f_\theta(P) || \frac{\partial^2}{\partial t \partial s} f_\theta(Q) \]
\[ = R_{i=0}^T \left( R_{j=1}^T v_t, s_j \right) || R_{i=0}^T \left( R_{j=1}^T v_t, s_j \right) \]
\[ = \left\{ \begin{array}{c}
\left[ t_0, t_1 \right] \ V_{p_0} \quad V_{Q_0} \quad V_{Q_0} \quad V_{Q_0} \\
\vdots \\
\left[ t_{k-1}, t_k \right] \ V_{p_{k-1}} \quad V_{Q_{k-1}} \quad V_{Q_{k-1}} \quad V_{Q_{k-1}} \\
\end{array} \right \} \]
\[ \text{(43)} \]
the operation is completed. A variety of interlocking and synchronization techniques, such as semaphores, mutual exclusions, and critical regions, have been proposed in real-time system techniques (McDermid, 1991).

**The Concurrent Process Relation**

**Definition 42.** The semantics of the concurrent relations of processes on $\Theta$, $\Theta(P \parallel || Q)$, is a double partial differential of the semantic function $f_\theta(P \parallel || Q)$ on the sets of variables $S$ and executing steps $T$, i.e.:

$$\Theta(P \parallel || Q) \triangleq \frac{\partial^2}{\partial t \partial s} f_\theta(P \parallel || Q)$$

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

$$= R \left( R_{vp} t, s \right) \int \int_{\Theta \left( \frac{\partial^2 T d}{\partial t \partial s} \frac{\partial^2 S d}{\partial t \partial s} \right)} R \left( R_{vp} t, s \right)$$

$$= R \int_{\Theta} R \left( R_{vp} t, s \right)$$

$$= \begin{cases} S_p \quad S_q \quad S_{pq} \quad \text{comRT} \\ t_0 \quad V_{dp} \quad V_{qp} \quad V_{imm} \\ (t_1, t_1) \quad V_{dp} \quad V_{pq} \quad V_{imm} \\ (t_2, t_1) \quad V_{dp} \quad V_{pq} \quad V_{imm} \end{cases}$$

where comRT is a set of interprocess communication variables that are used to synchronize $P$ and $Q$ executing on different machines based on independent system clocks.

The semantic diagram of the concurrent process relation as defined in Equation 44 is illustrated in Figure 13 on $\Theta(P \parallel || Q)$.

**The Interleave Process Relation**

**Definition 43.** The semantics of the interleave relations of processes on $\Theta$, $\Theta(P || || Q)$, is a double partial differential of the semantic function $f_\theta(P || || Q)$ on the sets of variables $S$ and executing steps $T$, i.e.:

$$\Theta(P || || Q) \triangleq \frac{\partial^2}{\partial t \partial s} f_\theta(P || || Q)$$

$$= R \left( R_{vp} t, s \right) \int \int_{\Theta \left( \frac{\partial^2 T d}{\partial t \partial s} \frac{\partial^2 S d}{\partial t \partial s} \right)} R \left( R_{vp} t, s \right)$$

$$= \begin{cases} S_p \quad S_q \quad S_{pq} \\ t_0 \quad V_{dp} \quad V_{pq} \quad V_{imm} \\ (t_1, t_1) \quad V_{dp} \quad V_{pq} \quad V_{imm} \\ (t_2, t_1) \quad V_{dp} \quad V_{pq} \quad V_{imm} \\ (t_3, t_1) \quad V_{dp} \quad V_{pq} \quad V_{imm} \end{cases}$$

(44)

Copyright © 2008, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
The semantic diagram of the interleave process relation as defined in Equation 45 is illustrated in Figure 14 on Θ( P ||| Q).

**The Pipeline Process Relation**

**Definition 44.** The semantics of the pipeline relations of processes on Θ, θ( P >> Q), is a double partial differential of the semantic function \( f_\theta(P >> Q) \) on the sets of variables S and executing steps T, i.e.:

\[
\theta(P >> Q) \triangleq \frac{\partial^2}{\partial t \partial s} f_\theta(P >> Q)
\]

\[
= \frac{\partial^2}{\partial t \partial s} f_\theta(P) \gg \frac{\partial^2}{\partial t \partial s} f_\theta(Q)
\]

\[
= R \bigg( R_{i=0}^{i\in \Theta(P>Q)} v_{t_j,s_j} \bigg)
\]

\[
= R \bigg( R_{i=0}^{i\in \Theta(P>Q)} v_{t_j,s_j} \bigg)
\]

\[
= \begin{bmatrix}
  t_0 & V_{0^P} & V_{0^PQ} & V_{0^Q} \\
  t_1 & V_{1^P} & V_{1^PQ} & V_{1^Q} \\
  t_2 & - & V_{2^PQ} & V_{2^Q}
\end{bmatrix}
\]

(46)

where \( S_P \) and \( S_Q \) denote a set of n one-to-one connections between the outputs of P and inputs of Q, respectively, as follows:

\[
\prod_{k=0}^{n-1} R(P_o(i) = Q(i))
\]

(47)

The semantic diagram of the pipeline process relation as defined in Equation 46 is illustrated in Figure 15 on Θ( P >> Q).

**The Interrupt Process Relation**

**Definition 45.** The semantics of the interrupt relations of processes, \( \theta( P \# Q) \), on a given semantic environment Θ is a double partial differential of the semantic function \( f_\theta(P \# Q) \) on the sets of variables S and executing steps T, i.e.:

\[
\theta(P \# Q) \triangleq \frac{\partial^2}{\partial t \partial s} f_\theta(P \# Q)
\]

\[
= \frac{\partial^2}{\partial t \partial s} f_\theta(P \# Q)
\]

\[
= R \left( R_{i=0}^{i\in \Theta(P>Q)} v_{t_j,s_j} \right)
\]

\[
= \begin{bmatrix}
  S_P & S_Q & \text{int}\Theta & \text{int}\Theta \\
  V_{0^P} & V_{0^PQ} & V_{0^Q} & \text{int}\Theta \\
  V_{1^P} & V_{1^PQ} & V_{1^Q} & \text{int}\Theta \\
  - & V_{2^PQ} & V_{2^Q} & \text{int}\Theta \\
  \text{int}\Theta & \text{int}\Theta & \text{int}\Theta & \text{int}\Theta
\end{bmatrix}
\]

(48)

The semantic diagram of the interrupt process relation as defined in Equation 48 is illustrated in Figure 16 on Θ( P \# Q), where
C(int'∩) and C'(int'∩) are the interrupt and interrupt-return points, respectively.

The deductive semantics of the three system dispatch process relations will be presented in the next section.

**DEDUCTIVE SEMANTICS OF SYSTEM-LEVEL PROCESSES OF RTPA**

The deductive semantics of systems at the top level of programs can be reduced onto a dispatch mechanism of a finite set of processes based on the mechanisms known as time, event, and interrupt. This section first describes the deductive semantics of the system process. Then, the three system dispatching processes will be formally modeled.

**The System Process**

**Definition 46.** The semantics of the system process $\hat{s}$ on $\Theta$, $\theta(\hat{s})$, is an abstract logical model of the executing platform with a set of parallel dispatched processes based on internal system clock, external events, and system interrupts, i.e.:

$$\Theta(\hat{s}) \triangleq \frac{\partial^2}{\partial t \partial s} f_0(\hat{s})$$

$$= \frac{\partial^2}{\partial t \partial s} f_0\{ n_{R,1}(\@ e_i \overset{s \leftarrow P}{\leftarrow} P_j) $$

$$\| (R_{\@ t_j \overset{TM \leftarrow P}{\leftarrow} P_j})$$

$$\| (R_{\@ int_i \overset{s \leftarrow P}{\leftarrow} P_k})$$

$$\}$$

$$= \prod_{SysShuntDown(tM \leftarrow f)} \{ n_{R,1}(R_{\@ e_i \overset{s \leftarrow P}{\leftarrow} P_j})$$

$$\| (R_{\@ t_j \overset{TM \leftarrow P}{\leftarrow} P_j})$$

$$\| (R_{\@ int_i \overset{s \leftarrow P}{\leftarrow} P_k})$$

$$\}$$

(49)

where the semantics of the parallel relations has been given in Definition 41, and those of the system dispatch processes will be described in the following subsections.

**The Time-Driven Dispatching Process Relation**

**Definition 47.** The semantics of the time-driven dispatching relations of processes on $\Theta$, $\theta(\@ t_j TM \leftarrow P_j)$, is a double partial differential of the semantic function $f_0(\@ t_j TM \leftarrow P_j)$ on the sets of variables $S$ and executing steps $T$, i.e.:
\[ \theta(@t, TM \downarrow t, P_1) = \frac{\partial^2}{\partial t \partial s} f_1(@t, TM \downarrow t, P_1) \]
\[ = \sum_{k=1}^{n} R(@t, TM \rightarrow \delta^2_{\delta t \delta s} f_1(P_k)) \]
\[ = \sum_{k=1}^{n} R(@t, TM \rightarrow \left( R_{tj} v_{r_j} (t_i, s_j) \right)) \]
\[ = \bullet(@t, TM \rightarrow R_{tj} v_{r_j} (t_i, s_j)) \]
\[ ... \]
\[ = \left\{ \begin{array}{c}
\@t, TM \\
S_{r_1} ... S_{r_n}
\end{array} \right\} \]
\[ \left\{ \begin{array}{c}
[t_1, t_n] \delta(@t, TM) \\
(t_1, t_n) \quad \@t \\
\vdots \quad \vdots \\
(t_1, t_n) \quad \@t \\
\end{array} \right\} \]
\[ = \left\{ \begin{array}{c}
S_{r_1} ... S_{r_n}
\end{array} \right\} \]
\[ \left\{ \begin{array}{c}
[t_1, t_n] \delta(@e, S) \\
(t_1, t_n) \quad \@e \\
\vdots \quad \vdots \\
(t_1, t_n) \quad \@e \\
\end{array} \right\} \]

where \( \bullet(@t, TM) = \bullet(@t, N) \) is the evaluation function as defined in Equation 17b.

The semantic diagram of the time-driven dispatching process relation as defined in Equation 50 is illustrated in Figure 17 on \( \Theta \).

The Event-Driven Dispatching Process Relation

Definition 48. The semantics of the event-driven dispatching relations of processes on \( \Theta \), \( \theta(@e, S \downarrow e, P) \), is a double partial differential of the semantic function \( f_0(@e, S \downarrow e, P) \) on the sets of variables \( S \) and executing steps \( T \), i.e.:
The semantic diagram of the interrupt-driven process relation as defined in Equation 52 is illustrated in Figure 19 on Θ.

CONCLUSION

Semantics plays an important role in cognitive informatics, computational linguistics, computing, and software engineering theories. 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 by the changes of statuses of a finite set of variables constituting the semantic environment of computing. Based on the mathematical models and architectural properties of programs at different composing levels, deductive models of software semantics, semantic environment, and semantic matrix have been formally defined. Properties of software semantics and relations between the software behavioral space and semantic environment have been discussed. Case studies on the deductive semantic rules of RTPA have been presented, which serve not only as a comprehensive paradigm, but also the verification of the expressive and analytic capacity of deductive semantics.

ACKNOWLEDGMENT

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 anonymous reviewers for their valuable comments and suggestions.

REFERENCES


Wang, Y. (2006b). On the informatics laws and deductive semantics of software, *IEEE Transac-


Yingxu Wang is professor of cognitive informatics and software engineering, director of International Center for Cognitive Informatics (ICCI), and director of Theoretical and Empirical Software Engineering Research Center (TESERC) at the University of Calgary. He received a PhD in software engineering from The Nottingham Trent University, UK, in 1997, and a BSc in electrical engineering from Shanghai Tiedao University in 1983. He was a visiting professor in the Computing Laboratory at Oxford University and Department of Computer Science at Stanford University during 1995 and 2008, respectively, and has been a full professor since 1994. He is the editor-in-chief of the International Journal of Cognitive Informatics and Natural Intelligence (IJCINI), and editor-in-chief of the CRC Book Series in Software Engineering. He has published over 300 journal and conference papers and 11 books in software engineering and cognitive informatics, and won dozens of research achievement, best paper, and teaching awards in the last 28 years, particularly the IBC 21st Century Award for Achievement “in recognition of outstanding contribution in the field of Cognitive Informatics and Software Science,” and the groundbreaking book on Software Engineering Foundations: A Software Science Perspective.