1

Introduction

1.1 The Aim of This Book

The theme of this book is to reconsider reflexivity as the essential property of sign systems. In this book, a *sign* is considered a *means of signification*, which at this point can be briefly understood as something that stands for something else. For example, a sequence of letters, 'computer', stands for an electronic machine. Such sign-based proxy relationships are powerful tools for communication. Signs function in the form of a *system* consisting of a relation among signs and their interpretations.

As will be seen further along in the book, a sign is essentially reflexive, with its signification articulated by the use of itself. Reflexivity is taken for granted, as the premise for a sign system such as natural languages. On the other hand, the inherent risk of unintelligibility of reflexivity, has been noted throughout human history in countless paradoxes. For example, in a Greek myth, Narcissus became immobile as a result of staring at his own reflection in the water. As shown by such examples, reflexivity has been an issue since ancient times in philosophy, logic, and language. Still, unless reflexivity causes contradiction or nonsensicality, reflexivity will stay as the manifest mechanism hiding the way in which sign systems work. Reflexivity becomes the theme mainly at the border between significance and insignificance. With artificial languages, however, it is necessary to design the border of significance and insignificance and thus their consideration will serve for highlighting the premise underlying signs and sign systems.

The artificial languages considered in this book are programming languages. They are artificial languages designed to control machines. The problems underlying programming languages are fundamentally related to reflexivity, and it is not too far-fetched to say that the history of programming language development is the quest for a proper handling of reflexivity. This book does not merely survey the consequences, but attempts to consider programming languages from a broader viewpoint of signs and sign systems in general. The domain serving this purpose is *semiotics*, the theoretical framework in which the general properties of signs and sign systems are described. In particular, the aim of the book is to consider the nature of signs and sign systems through discussion of programming languages by semiotics.

Such an endeavor highlights the difference between computer signs and human signs. The comparison of machines and humans is a recurring theme in conversations in daily life, in science fiction, and in philosophical discussions in academic domains, despite the fact that humans and machines are utterly different, with one being biological and the other mechanical. Countless metaphors compare computers to humans, and vice versa. This very fact suggests machine-based and human systems can be considered similar, to some extent.

A common test bed applied in this book regarding this similarity is the sign systems. Readers might doubt the plausibility of this comparison of human and computer sign systems. The world of computer software indeed consists only of signs, because all information processed on computers is ultimately made up of zeros and ones. Still, computer languages often appear very limited as compared to human language, especially to those who are not programmers. This delineation of human and computer language, however, is not as trivial as a division based on human language being complex and computer language being simple; rather, the delineation is a difficult issue, as seen in controversies over formal theories of language. At the same time, some readers might also wonder to what extent humanity can be considered merely in terms of signs and sign systems. Such an approach, however, is indeed extant in the humanities, particularly in semiotics, linguistics, and philosophy. It is therefore not an oversimplification to compare human language and computer language on the common test bed of sign systems.

Considering both as sign systems, their comparison seems to lead to highlight the premise upon which our sign system is founded. Namely, the application of semiotic theories to programming enables the consideration, in a coherent manner, of the universal and specific nature of signs in machine and human systems (see Figure 1.1). Such a comparison invokes the nature of descriptions made by humans in general, of the kinds of features a description possesses, and of the limitations to which a description is subject. These limitations, this book suggests, are governed by reflexivity. Moreover, the difference between computer signs and human signs lies in their differing capability to handle reflexivity, which informs both the potential and

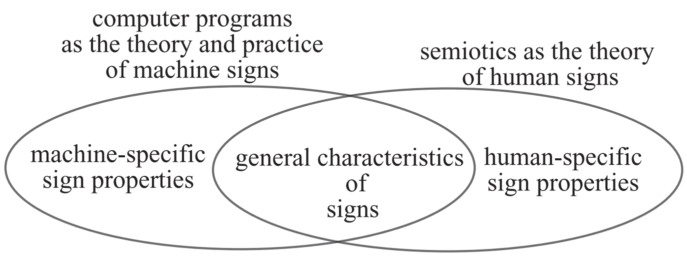
computer programs as the theory and practice of machine signs
semiotics as the theory of human signs
machine-specific sign properties
general characteristics of signs
human-specific sign properties

FIGURE 1.1. Human signs and machine signs.

the limitations of the two sign systems. While people do get puzzled, they seldom become uncontrollable because of one self-reference. In contrast, for computers, reflexivity is one of the most frequent sources of malfunction.

Given this key concept of reflexivity, this book straddles the domains of semiotics and computation, as well as those of the humanities and engineering, and the study of machines and humans. Such an interdisciplinary approach can be expected to bring forth contributions to each participating realm.

1.2 Computational Contributions to Semiotics

The purpose of semiotics, in general, is to explain signs and sign systems, to describe their general characteristics and structure, and thus to establish a methodology for their explanation. Behind these considerations is the question of the meaning of meaning, or how signs convey meaning. The domain of modern semiotics was established by Saussure and Peirce, with roots dating back to the ancient Greeks and the scholastics. Before the nineteenth century, the distinction between the arts and sciences was not recognized, but scholars have since subdivided academia in this way. As a consequence, the semiotic discipline was initially developed mainly by scholars in the arts, and the preponderance of the existing literature on semiotics is humanities oriented.

The object of semiotic analysis has traditionally been sign systems for human interpretation: natural language, texts, communication, codes, symbolic systems, sign language, and various art forms. There are some exceptions, such as biosemiotics, but the majority of studies examine human communication. Unfortunately, this means that semiotic studies have normally been conducted without a clearly delineated separation between the sign system to be analyzed and that used for its study. As a consequence, a solid theoretical foundation for semiotics has yet to be established. For example, even considering models of signs alone, various philosophers have each presented their own ideas, and how these ideas correspond has remained unclear.

The use of computer languages as a semiotic subject does not suffer from this failing, however, because human beings do not think in machine language. Computer languages have their own type of interpretive system, external to the interpretive systems of natural languages. All computer language expressions are meant for interpretation on machines. Here we may remark how computer programs have marked characteristics in terms of possessing explicit, external interpretation schemes. They are open and explicit only for artificial notations, and this holds for other artificial notational systems, such as those in mathematics, logic, and even music and dance. Still, these notations are meant for human beings to interpret. For example, mathematics is another well-formed, rigorous sign system, but it is ultimately interpreted by human beings, and therefore certain repetitive elements are frequently omitted.

In contrast, every computational procedure must be exhaustively described in a programming language, inclusive of all repetitions, so that the processing comes to a halt and is valid for the intended purpose. With respect to interpretation, in a sense, there is in theory no system better than that of computer languages because they are truly formal, external, and complete in scope. Since this interpretation is performed mechanically, it is explicit, well formed, and rigorous. Computer languages are the only existing large-scale sign system with an explicit, fully characterized interpreter external to the human interpretive system. Therefore, the application of semiotics to computer languages can contribute, albeit in a limited manner, to the fundamental theory of semiotics.

Regarding this point, the reader might object that computation is quite different from human interpretation, which is true. Indeed, as we shall see in more detail in Chapter 4, computation is equivalent to chains of mere substitutions. Substitution, however, is a simple scheme that could also be considered a fundamental procedure underlying human interpretation. Moreover, even though the interpretation of artificial languages differs from that of natural languages, its study can bring forth a better understanding of interpretation in general, as compared with neglecting such an approach.

Consequently, applying semiotics to the external, rigorous system of computer programs helps formalize certain aspects of the fundamental framework of semiotics. Understanding the semiotic problems in programming languages leads us to formally reconsider the essential problems of signs. Such reconsideration of the fundamentals of semiotics could ultimately lead to an improved and renewed understanding of human signs as well.

1.3 Semiotic Contributions to Computing

Computers are now indispensable in our daily lives. Behind every computational system is a program. Programming languages are among the most widely applied artificial languages. People have attempted to describe various conceivable phenomena in terms of computer programs, and the only form of language exceeding this coverage is natural language. Computer programming is thus the most successful application of artificial language, and the scale of its practice is vast.

Most programs are generated by human beings. As expressions written in programming languages are interpreted by both humans and machines, these languages reflect the linguistic behaviors of both. The history of the development of computer languages can be characterized as the history of an effort to transform a mere mechanical command chain into more human-friendly expressions. Thus, an analysis of recent, well-developed programming languages may reveal significant aspects of human linguistic behavior.

Many of the concepts, principles, and notions of computer programming, however, have derived from technological needs, without being situated within the broader context of human thought. An example is that the paradigm of object-oriented programming is considered to have been invented in the 1960s. This was, however, no more than the rediscovery of another way to look at signs. The technological development of programming languages has thus been a rediscovery of ways to exploit the nature of signs that had already been present in human thought.

The application of semiotics to programming languages therefore helps situate certain technological phenomena within a humanities framework. To the extent that computer programs are formed of signs, they are subject to the properties of signs in general, which is the theme of this book. That is, the problems existing in sign systems generally also appear in programming languages. This semiotic discussion of programming theory has sought to provide a possible explanation of why programming and current computer systems have necessarily converged to their current status and how computation is fundamentally related to human semiosis.

1.4 Related Work

Generally, the relation between signs and reflexivity, without noting specific examples at this juncture, has been a recurring theme appearing in many investigations, in which at least two broad genres are apparent. First is the reflexivity existing in natural languages and in human knowledge based upon natural language. Second is the reflexivity existing in a specific and formal language, such as in mathematics or logic. As seen in these domains, the former was developed mainly in the humanities, whereas the latter belongs more to science and engineering. In these two major scholarly domains, there have been similar considerations. This book addresses the theme of the reflexivity of signs but attempts to bridge the two genres of natural and formal language to situate reflexivity as the general property of signs and sign systems.

From the more specific viewpoint of semiotics in computing, in recent decades, there has been evidence of a growing interest in semiotics on the part of those concerned with computer science and on the part of those applying information technology to the domain of the humanities. The earliest mention of this topic was a brief four-page article in *Communications of the ACM* (Zemanek, 1966), which emphasized the importance of the semiotic analysis of programming languages. Publication of an actual study analyzing the computing domain, however, had to wait until publication of studies by Andersen (1997) and Andersen, Holmqvist, and Jensen (1993, 2007). Their work modeled computer signs within information technology in general. Such work was important because it opened the domain of the semiotic analysis of computing, and it has been continued further by authors such as Liu (2000). Ever since then, this domain has progressed through papers in Walter de Gruyter's *Semiotica* and the *Journal of Applied Semiotics*, through conference/workshop papers on *Organizational Semiotics*, and also through Springer's *Journal of Minds and Machines*, which takes a more philosophical approach. Other related publications are those of Floridi (1999, 2004), which provide wide-ranging discussion of philosophy as applied to the computing domain. In terms of application, the most advanced domain in this area of semiotics is human–computer interaction, the advances in which have been elucidated in a book by de Souza (2006).

The appearance of all these titles and papers would seem to suggest a growing interest in books relating to the technological application of the humanities. Nevertheless, in this domain much more remains to be done in exploring the relationship between computing and semiotics. This book is specifically concerned with semiotics in relation to programming languages, which represents an area of computing not yet covered in great detail by any existing book on computational semiotics.

1.5 The Structure of This Book

The fundamentals of semiotics can be examined from three viewpoints: first, through *models* of signs as an answer to the question of what signs are; second, through *kinds* of signs and the content that signs represent; and third, through *systems* of signs constructed by those signs. Accordingly, the main part of this book is organized into three corresponding parts. Part I starts by formalizing models of signs through consideration of the correspondences between two large trends in sign models. Since each trend incorporates the notion of kinds of signs and entities, Part II considers these trends' correspondences, based on Part I, and how they appear in computer programs. Finally, Part III compares and contrasts human and machine systems in terms of the foundations established in Parts I and II.

This structure differs from studies on language in general, which are usually organized according to the domains of syntax, semantics, and pragmatics. I chose the organization described above because the approach of this book arises from semiotics, which is situated at the most fundamental level of language, even before considering elements of linguistic structure such as syntax. The levels of syntax, semantics, and pragmatics do appear in the book but are distributed throughout the various chapters at appropriate points, when necessary. In this sense, the term *language* in this book does not signify a language in the context of linguistics, that is, a system with morphology, syntax, semantics, and pragmatics. The signification of language in this book is in its most abstract form, referring to a kind of sign system in which the signs are linguistic elements. In other words, I treat a language as a relation among linguistic elements and their interpretations.

Additional guidance on the structure of this book is provided in Figure 3.8 in Section 3.5, which shows a map of the chapters, indicating which part of the sign system is the focus of each chapter. This map appears after the definitions of basic concepts and terms in Chapters 2 and 3.

As mentioned previously, this book follows an interdisciplinary approach covering both semiotics and computer programming. Both domains require a degree of expertise, and for this reason an introduction to each is needed. An interdisciplinary book such as this one usually has several chapters of introductory material followed by the main content. This book, however, does not include such introductory chapters in either semiotics or computer programming. Rather, introductory material is provided throughout as needed.

For semiotics, the reason for not providing introductory material at the start is that I have not simply taken a theory established by just one semiotician and applied it to computer programs. When I began writing this book, semiotic theory was not sufficiently established to be straightforwardly applied in a complete form that could be introduced at the beginning of the book. Application of semiotic theory to a well-formed corpus required dismantling, reconsidering, and reconstituting the constituent theories. Most of the chapters in this book treat a semiotic problem that I find fundamental, and the problem is analyzed and hypothetically solved by some adaptation of semiotic theory through its application to computer programs. These hypothetical conclusions currently apply, in the most rigorous sense, only to computer programs. To show the potential of these conclusions, however, they are also applied to the artwork at the beginning of each chapter, thus offering an intuitive or metaphorical introduction to the hypothetical problem explored in the chapter. Although I am no more than an amateur with regard to the fine arts, these parts are included in the hopes of helping the reader intuitively grasp the significance of each chapter and to make the book more interesting.

In contrast, for programming languages, I refer only to theories and concepts already extant within the computer programming domain and merely utilize them for semiotic analysis: since a programming language is well-formed and rigorous, the relevant theory is fundamentally clear. Where necessary, introductions to a programming language and its related theoretical material are made at certain points in the book to clarify the point of an argument. For example, the next chapter introduces two programs and explains the purposes of the programs to an extent sufficient to fulfill the purpose of the chapter, which is to define what I mean by computer signs. A more substantial explanation of the underlying concepts of the two programming languages introduced in the next chapter is given in Chapter 3, along with a semiotic interpretation of these languages' differences. This is by design because this book considers various programming languages: each chapter is based on specific programming languages that best highlight the point of the argument. A thorough introduction to each language would itself require an entirely separate book. Among numerous programming languages, the two representative ones introduced here are Haskell (Chapters 3, 7, and 10) and Java (Chapters 3, 5, and 6). Moreover, Chapter 4 is based on the lambda calculus and other languages appear here and there throughout the book. Since there are many good books about each of the languages appearing here, readers interested in a more thorough understanding of these languages are invited to refer to these additional sources.

Both semiotics and computer programming have their own technical terms; these terms sometimes overlap, making the situation more complex. For example, the term *argument* can mean the persuasive thrust of a discussion in general, but in semiotics it often signifies a Peircian argument, whereas in computer science it refers to a parameter given to a function. To clarify such terminology, a glossary providing basic definitions, concepts, and the reasoning behind certain of my lexical choices is included at the end of the book, with separate term lists for both semiotics and programming. The usage throughout the book follows the definitions given in the Glossary.

The notations used in this book are as follows. Executable program code is shown in `typewriter face`, whereas mathematical notations, titles, emphases and important terms are in *italics*. Sample terms and phrases appearing in the book are enclosed in single quotation marks, whereas inline quotes taken from other references are enclosed in double quotation marks. Strings appearing in programs are enclosed in double quotation marks, and parentheses within program code are sometimes added, even when unnecessary, to explicitly indicate a program's composition. Some references have detailed information, such as chapters, page numbers, and paragraphs, indicated within square brackets. The format of this information differs according to the reference: for example, the format for Peirce (1931) is the composite of two numbers, such as [2.345], where the first number indicates the volume and the latter number gives the section number in the *Collected Papers* published by the Harvard University Press. I do not explain how to read the reference format for every reference; interested readers can consult the referred documents.

The individual chapters are based on my papers published in Walter de Gruyter's *Semiotica*, in the *Journal of Applied Semiotics*, and in the *Journal of Minds and Machines*. Specifically, for Part I, the original basis of Chapter 3 appeared in Tanaka-Ishii (2006), that of Chapter 4 in Tanaka-Ishii and Ishii (2008b), and that of Chapter 5 in Tanaka-Ishii and Ishii (2008a); for Part II, Chapter 6 appeared in Tanaka-Ishii and Ishii (2007), Chapter 7 in Tanaka-Ishii and Ishii (2006), and Chapter 8 in Tanaka-Ishii (2009); and for Part III, Chapter 9 appeared in Tanaka-Ishii (2008) and Chapter 11 in Tanaka-Ishii (2010). The content of these journal papers was modified for the purpose of archiving and also elaborated to make the overall arguments consistent.

2

Computer Signs in Programs

2.1 Introduction

A programming (or computer) language is an artificial language designed to control computers and machines. A text written in a programming language is called a program, and machines are thus controlled using programs. Programming languages follow strict rules on syntax and semantics, and a programmer must follow these rules to generate a program with the expected behavior. Once written, the program text is syntactically analyzed, optimized, or compiled if necessary, depending on the language, and then it is executed, or run, on machines.

Since the major theme of this book concerns signs, this chapter introduces the nature of the computer signs employed in programs before embarking upon the primary focus of the book in the next chapter. Recent programming languages are highly developed and have many distinct features and functions. A proper formal introduction to these programming languages would therefore require a book for each language. Because this book considers different languages in parallel, a thorough introduction to these languages is beyond the scope of the book. This chapter therefore limits the introduction of programming languages to the extent that is necessary to proceed to the main part of the book.

The introduction is briefly made through two comparable executable example programs written in two different programming languages. From among the substantial number of different programming languages, Haskell (Bird, 1998) and Java (Arnold *et al.*, 2000) were chosen because these languages represent two paradigms – a functional language and an object-oriented language – that have interesting features from a semiotic viewpoint. These two examples represent the essences of the languages needed for semiotic