From Reflection to Interaction: An Indirect Approach to the Philosophy of Computation*

Hiroyuki Miyoshi†
Department of Computer Science
Kyoto Sangyo University

September 19, 2003

1 What Is Computation?

As computers and networks are pervading in our daily life, The word 'computation' has become popular not only in natural science but also human and social sciences. However, the use of the word is not necessarily the same in these areas. What modern computers do is indeed far beyond simple computation. It is not easy to clearly distinguish computation from other phenomena. Therefore it is significant to question afresh what computation is. Then, how to approach the question? At least, to try to answer it directly, it is necessary to describe computation. But it is impossible in at least two senses, as follows.

For example, consider describing computation progressing now in my computer with which I am writing this paper. I may spill coffee on the keyboard and cosmic ray may change a bit on memory. It is impossible to completely describe all about computation for the same reason as the frame problem. An individual which computes is also indescribable. For example, because my computer needs electricity, it is vague whether it includes electric transformers and generators. This is the indescribability of tokens of computation. On the other hand, consider more abstract computational models. As an individual to compute, a Turing Machine is described as a tuple of a set of finite characters, a finite set of states including an initial state and final states, and a set of rules to change the state, rewrite the character on the head, and move the head left or right; a computation done by a Turing Machine is specified as a finite initial sequence of characters on the tape. But these are just descriptions and their semantics must be given. If the semantics of a description is to be provided

*This paper has been revised based on the Japanese preliminary version and contains improvements for English-speaking readers.

†E-mail: hxm@cc.kyoto-su.ac.jp.

by another description repeatedly, it comes into an infinite regress. This is the indescribability of type of computation.

These indecribabilities show that neither of a simple epistemological treatment and an ontological one is adequate for the question of what computation is. Thus in this paper, instead of direct approaches, we introduce a suite of metaphysical devices built on studies in computer science. Roughly speaking, we introduce the mode of phenomenals as an ontological device and therapeutic understanding as an epistemological one and find that they alternately depend on each other. We adopt the strategy to prompt readers to understand descriptions of those devices on the fact that this theory is applicable to understanding the descriptions of the theory itself. In other words, we dare to describe something transcendent and instead loosen the premise that everyone can understand the description. By exploring this route, we aim to smoothly connect between natural, human, and social sciences. Furthermore, it is desirable that the theory does not contain factors specific to human beings. Once we construct metaphysics on them, we have to confront the hard question of what human beings are. The theory we propose here is not restricted to particular entities and scales.

2 Computational Reflection

As the first step to consider the design of such devices, we pick up computational processes executing a program where the behavior of the system which execute the program can be also changed by the program itself. This kind of computation seems to be very special but it has been used for a long time. For example, in early days, the technique of rewriting a running program was used to reduce consumption of poor memory (of course, it is dangerous). In operating systems, several interfaces have been prepared for process to control the behavior of the systems such as process scheduling or power-off. In object-oriented programming systems like Smalltalk-80, objects can change the behavior of their system by sending messages to those objects which are part of the system.

In the context of programming languages, however, Brian C. Smith first proposed a unified approach to such kind of computation [Sm82, Sm84]. He analyzed in depth the design of programming languages and self-reference occurring in their execution. From this experience, he extracted the idea of **(computational) reflection**. Furthermore, by designing a concrete programming language, 3-Lisp, and writing an interpreter of it, he showed that reflection is actually implementable on computers. In this section, we summarize the Smith's approach, on which we develope metaphysical considerations in the following sections.

2.1 2-Lisp and 3-Lisp

Introduction of reflection by Smith is divided into two steps. The first one is to 'rationalize' the Lisp programming language, or redesign it from the two

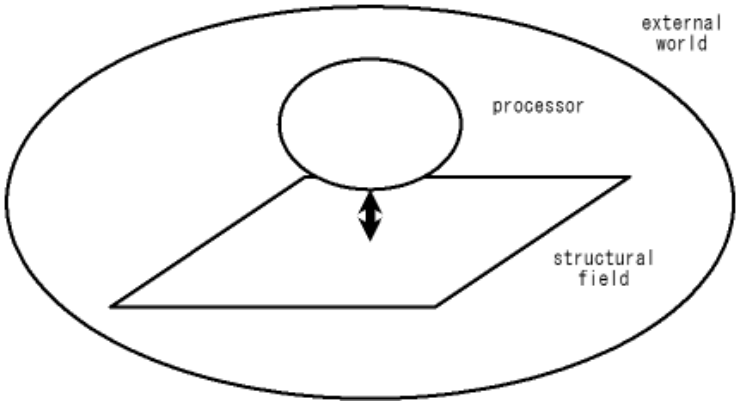

Figure 1: A Processor and a Structural Field

viewpoints of reference and execution into the language 2-Lisp (the second one is explained later). For this step, we introduce a model consisting of a **processor**, a **structural field** and the **external world**, which respects our simple image of execution of programs (Figure 1). In this model, execution of a program is regarded as processing of contents of the structural field by the processor. While the structural field is an abstraction of memory which stores programs and data of Lisp, **structures** are elements of the structural field which are abstractions of cell structures on memory implementing both programs and data structures. Structures may refer to other internal structures or external objects such as natural numbers.

The important observation is that it is interaction between the processor and the structural field that make a structure in the structural field work as a reference. A referential structure is by itself nothing but a structure. In real programming, the semantics of pointers (e.g. address values in memory) is given by the system who traces them.

Then how does that interaction happen? The processor in our model interacting with the structural field is also implemented by a mechanism. Applying the model to the processor, we can think of it as implemented by another pair of a processor and a structural field (e.g. as an interpreter). We can endlessly repeat such an application. Consequently, this type of description of computation, called the operational semantics, results in a sort of infinite regress. We usually stop this process at a point where the semantics becomes clear enough. But for our purpose to answer the question stated above, we cannot be satisfied with it.

Then we consider a **metacircular interpreter** which is an interpreter of a language written in the language itself. (It is executable if we already have another interpreter of the language). In this case, if we know the semantics of one language, we can regard the program of the metacircular interpreter as the whole description of the system. Note that the semantics of the language is not

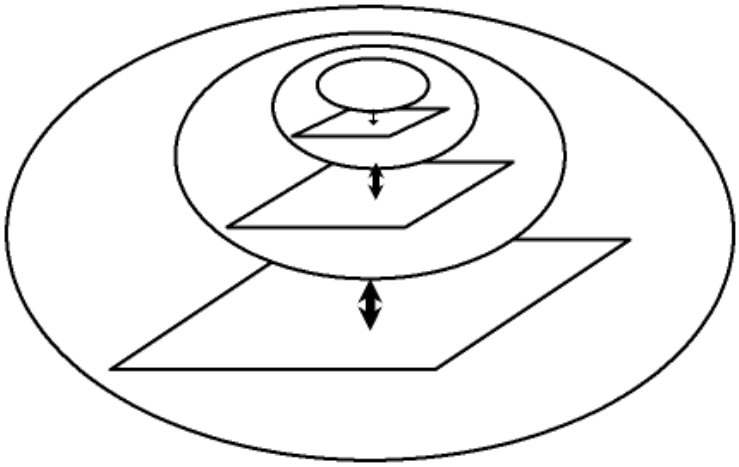

Figure 2: A Metacircular Processor

unique (for example, the semantics collapsed to the void is also valid). When we understand the semantics, we construct it by guessing, checking and revising it with our present knowledge and the fact that the interpreting language and the implementation language are the same. It may be done for an instant or take very long time. Applying the idea of a metacircular interpreter to our model, we obtain a **metacircular processor** which is an infinite tower of processors and structural fields. Note that each layer of the tower is independent and has no communication with other layers except that one layer is implemented by the upper layer. Therefore such an infinite tower is not necessary to give a semantics to the language; ordinary methods for semantics are also straightforwardly applicable. But to introduce self-modification of behavior of a system as communications between layers, the tower is essential.

As the second step of Smith's introduction of reflection, to make such communications possible, we add to 2-Lisp a mechanism to define reflective functions which is executed in the upper layer and returns the result to the present layer. This language is 3-Lisp, which enables us to write a program to redefine even the semantics of language constructs by extracting and installing data of the upper layer about current execution at the present layer. In general, this sort of mechanisms is called **(computational) reflection** and the infinite tower in our model is also called the **reflective tower**. In reflective systems, a self-representation of a computational process is provided to the process itself (e.g. in 3-Lisp, the self-representation is a pair of the environment and the continuation of current execution of the interpreter program at the upper layer). For a computational process executing a program to refer and modify its self-representation may causes changes of the semantics of the program (**causal connections**). Thus, we have a recipe to make a reflective system: a metacircular interpreter + causal connection = reflection.

3 Toward the Philosophy of Computation

Reflective computation explained above makes several points explicit which is not clear in the case of ordinary computation. In this section, we show the philosophical ideas which come from them.

3.1 Causal Transcendence and Institutional Existence

A reflective computational process can control the process itself. The boundary of such a process is complicated and difficult to be determined because reflection breaks a nest of processors. Rather, like a metacircular interpreter, it is natural to think of a description of the process (e.g. a program) itself as the boundary. However, considering that the semantics of structures is provided by interaction between a processor and a structural field, descriptions are deficient to properly treat actual entities in computation.

Extending these thoughts from computation to phenomena, we reach the following ideas. For one entity to effectively exist, it requires another entity who recognize or understand its description, or its boundary. However, for that entity to effectively exist again, the description of that has also to be recognized by the third one. This infinite regress shows that effective existence of entities always needs something transcending descriptions. (We refer to descriptions here in a broad sense; we might be able to rephrase them as a sort of patterns). We call this situation of entities **causal transcendence**. The undescribability of types of computation mentioned at the beginning says that for programs to be executed, they need causal transcendence. Moreover, because descriptions is retained in causal transcendence, we can see that when entities are recognized, their descriptions provide reality over a portion of the transcendential duration. We call this situation of entities **institutional existence** and for descriptions to give identities to entities as above **institutional cut**. The undescribability of tokens of computation says that computers as subjects of computation are institutional existences and their descriptions are contiguous to something transcendent. [1]

3.2 Interactions and Their Descriptions

Though in our model for 2-Lisp, a processor and a structural field are of different sorts and interaction between them is asymmetric, the structural field is also to be realized by some mechanisms like the processor. In fact, memory chips as well as CPUs are semiconductor devices. Thus it is natural to primarily think of the interaction as symmetric between the same kind of entities.

Furthermore, the boundary of entities is not primarily given but established by description of interaction between those entities, and the description is also provided by another interaction. Therefore we have to simultaneously consider

[1] This conception is inspired from [Ic01] and terms are also borrowed from it. However, it strongly emphasizes the concept of personality, which is different from our idea.

both interaction between entities as described individuals and individuality of entities through descriptions generated by interaction.

3.3 Therapeutic Understanding

Though reflective system is that which can modify its own behavior, the range of modification is restricted by description in a particular form of causal connection so that we can understand the whole system. If we extend the range, we have to change the form itself. Then we have to specify an extension of the form by descriptions in another form. Here also an infinite regress occurs[2].

To cope with this difficulty, we choose another approach inspired from metacircular interpreters. That is, we introduce a set of forms for description in which both desriptions and something transcending descriptions (called duration) appear symmetrically. Then the fact that the forms are also applicable to the situation that we are to understand descriptions in the forms, provides our understanding of what descriptions and duration in the forms are. This mode of understanding is named **therapeutic understanding**. An example is your understanding when you understand that the sentence "You are reading this sentence" is true. This mode provides so-called obvious understanding and the obviousness itself is also understood in this mode[3]. This resembles Descartes' understanding of cogito. But it is different from the mode above in several points that we do not require his method of hyperbolic doubt and this mode does not need to be conscious of it and that we also positively consider endless processes of understanding. Then obviousness has its own intensity and it can be lost.

4 Hume-Bergson Form

4.1 Description, Duration, and Phenomenals

In this section, we propose a suite of conceptual devices based on the previous sections. Firstly, we introduce a set of graphically-presented forms, **Hume-Bergson Form** (HBF). Figure 3 is the first form (HBF-1) of them. The upper triangle on the diagonal line is the region of **description** and the lower one under the diagonal is the region of **duration**. The left side of the rectangle denote the **pure description**, and the right side the **pure duration**. Any actual **entity** is denoted as a vertical line between the pure description and the pure duration. Entities include both object-like ones (e.g. computers) and process-like ones (e.g. computations). It is important that we give no primary distinction between them. Because the word 'entity' strongly suggests the object-like property, we call it a **phenomenal** instead. The fact that the vertical line necessarily

[2]Recently Irifuji stated in [Ir01] that this relativity reaches a kind of absolutness. But because his approach cannot be a direct one, a jump somewhere in the regressions is implicitly required.

[3]Once the metaphysical devices presented here are understood therapeutically and get obvious, the description of this paper will be thrown away.

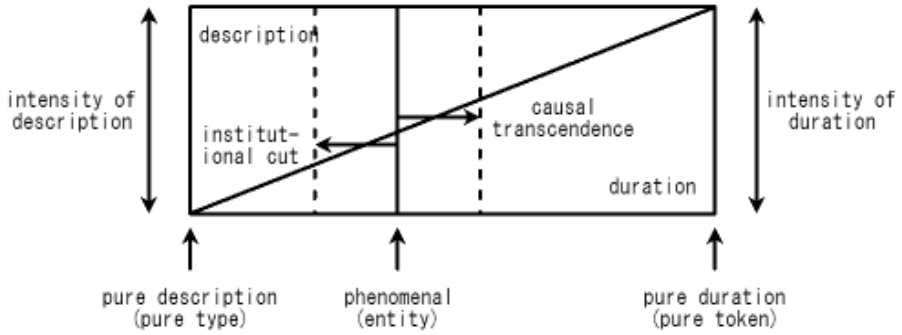

Figure 3: A Phenomenal in Hume-Bergson Form (HBF-1)

crosses the diagonal represents that any description of an entity is supported by some duration. That is, phenomenals are contributed by both description and duration. **Causal transcendence** and **institutional cut** are respectively right and left transitions of a phenomenal in HBF-1.

Indeed, on the one hand, the Humean view of entities is primarily interested in descriptions of cusality. But to recognize or understand them, we need 'impression' transcending them [Ic01]. On the other hand, the Bergsonian view stresses that the duration is more primitive than descriptions. But it is recognized by 'intuition', which enables us to treat it by descriptions [Be34]. Each of them needs the counterpart. This is the reason why we combine the names of Hume and Bergson.

Note that the pure description only institutionally exists and the pure duration does institutionally not exist. The former cannot be effective because without duration, it cannot concern any causal transcendence; the latter cannot be treated because it does not have any institutional description to be detected. In these senses, both of the left and right ends in HBF-1 are virtual and any entity is situated between them.

4.2 Representation of Interactions

It is not easy to analyze an interaction as a phenomenal in HBF-1. To represent a phenomenal as an interaction between several individual phenomenals, we need to express their boundaries which are also to be descriptions. Therefore we introduce the second form (HBF-2) which is a cross section of HBF-1 at a vertical line corresponding to the interaction (Figure 4). The broken line in this form corresponds to the intersecting point of the diagonal and the vertical line in HBF-1, the area over which is the region of description and that under which is the region of duration. Interactions in description and duration are denoted respectively by a solid double-headed arrows and dotted ones. Accordingly, the two transitions of interactions, i.e. causal transcendence and institutional cut, can be represented as disappearance and appearance of description (Figure 5).

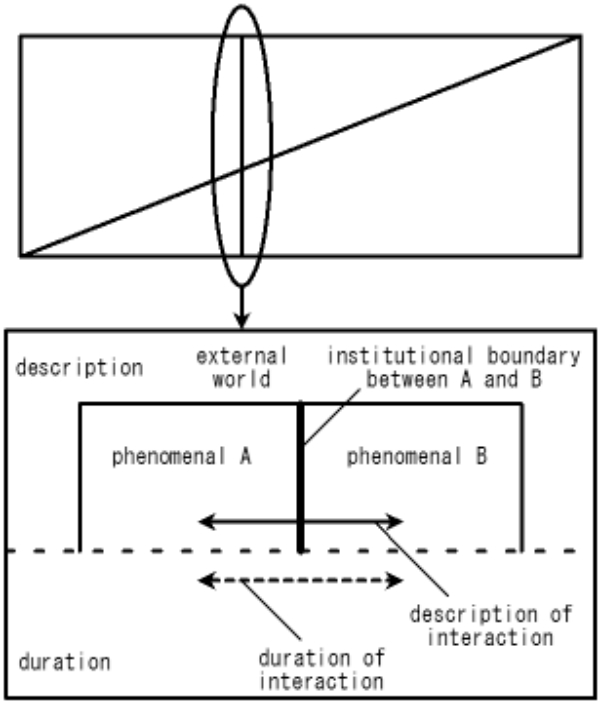

Figure 4: An Interaction in Hume-Bergson Form (HBF-2)

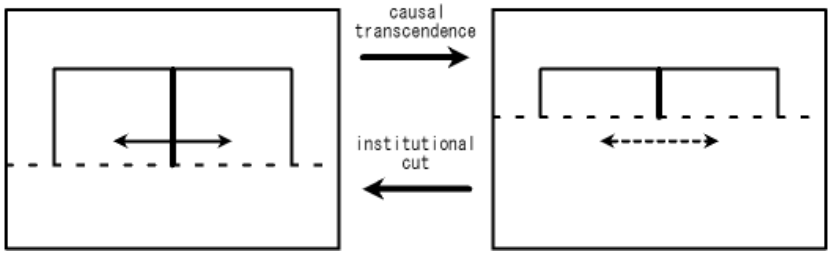

Figure 5: Causal Transcendence and Institutional Cut

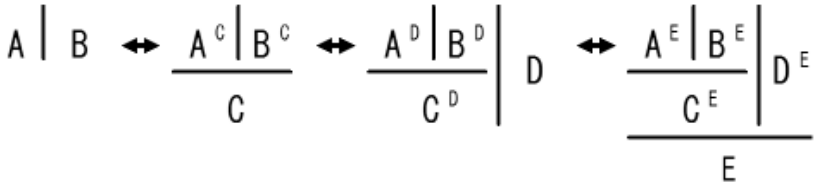

Figure 6: Descriptions Caused by Interaction (HBF-3)

However, it is important that these transitions in HBF are **not** in the described flow of time because it should be observed by other entities and hence time as such is institutional in HBF.

4.3 Emergence of Descriptions

Moreover, (descriptions of) interacting phenomenals A and B in HBF-2 are not primarily exist but appear through interaction of another phenomenal C with the (phenomenal of) interaction between A and B. If necessary, we write A^C and B^C to emphasize it. This can be extended with more phenomenals D, E, $\cdots$. Thus we introduce the third form (HBF-3) to describe that situation (Figure 6). As a matter of convenience, we also use such literal expressions as $(A^C \,|\, B^C) \,|\, C$. At first glance, this seems to be an infinite regress. But as explained above, it heppens only when repeatedly describing interactions which provide other descriptions. Unless starting questioning them (that is, therapeutic understanding, temporary acceptance of a description, or describing nothing), it is not vicious. Another important point of HBF-3 is that, for example, B can be also C in Figure 6. In other words, B can be connected with C through duration or description. From the view explained here, this is a primitive mechanism to generate complicated self-descriptions in various levels.

As an example, we express reflection in HBF-2. As stated above, for a processor to read and modify a structural field is interactions through descriptions. Then a metacircular interpreter is represented as a nesting structure in which each boundary between layers is the description of interpreter (Figure 7(a)). Reflective facility is enabled by adding to each layer causal connection to control its upper layer (Figure 7(b)).

5 Concluding Remarks

In this paper, we propose a metaphysical theory to treat computation properly. How "good" is this? Many of philosophical arguements concerning computation, e.g. those in Artificial Intelligence or Artificial Life, do not reexamine the notion of computation itself, which often causes to make problems difficult to solve. Our strategy is that we set up a general framework and, by considering problems therein, eliminate such difficulty of them. However, because even the notion of computation is disappeared, we have to reconstruct it if necessary. In this

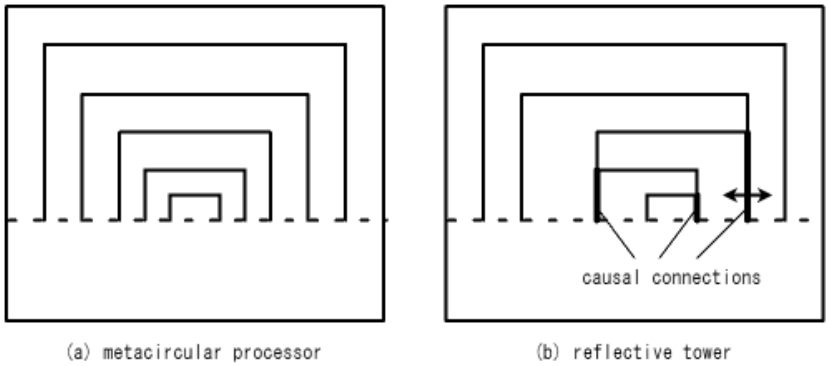

Figure 7: Reflection in HBF

sense, the theory may be too general. Hence, as the next step, it is necessary to apply the theory to various instances and refine it for each case to show its flexibility and availability. Such a theory is not directly verified but justified by having many applications or inspiring us with many ideas. Works in progress concerning the theory are as follows:

- Philosophy of becoming
  - Symmetric presentation of Deleuze's difference and Derrida's differance of in HBF
  - Spinoza's unity and multiplicity of substance as the mode of phenomenals
  - Nietzsche's eternal recurrence as therapeutic understanding of therapeutic understanding
  - Unification of clocked time and time of becoming
  - Comparison with Deleuze and Nishida
- Ethics and mind
  - Psychiatry (depersonalization and delusion of control in HBF)
  - Conciousness studies (qualia as phenomenals)
  - Structures of self and others, privateness and publicness (Wittgenstein, Kripke, communication theory)
  - Foundations of ethics (including Nietzsche's one)
  - Mediation between bioethics and biology
- Mathematics, computation, and philosophy of science
  - Mathematics and computation of individuals (formal ontology, formal concept analysis, mereotopology, morphology, domain theory of solid models, etc.)

– Partial representation of interaction as higher-dimensional categories
– Wittgenstein's foundation of mathematics, mathematical obviousness as therapeutic understanding
– Foundations of android epestemology and computational philosophy of science

Acknowledgements

The author thanks those people who gave him valuable comments to his talks and drafts on this subject and, among others, the members of the "Philosophy of Computation" project. This research was partially supported by the Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (B), 2001-2, 13480115, Exploratory Research, 2002, 14658096, and Scientific Research (B), 2003, 15320007.

References

[Be34] H. Bergson, *La Pansée et le Mouvant*, 1934.

[Ic01] M. Ichinose, *Gen'in to Kekka no Meikyu (The Labyrinth of Cause and Effect)*, Keiso Shobo, 2001 (in Japanese).

[Ir01] M. Irifuji, *Soutaishugi no Kyokuhoku (The Ultima Thule of Relativism)*, Shunjusha, 2001 (in Japanese).

[Sm82] B. C. Smith, *Reflection and Semantics in a Procedural Language*, Ph.D Thesis, MIT Laboratory for Computer Science Report MIT-TR-272, 1982.

[Sm84] B. C. Smith, "Reflection and Semantics in Lisp", in: *Conference Record of the Eleventh Annual ACM Symposium on Principles of Programming Languages*, pp.23–35, 1984.