Semantics of recursive programs has been extensively studied for more than 30 years, and now there exist several well-established theories, most notably the fixed-point semantics and domain theory developed by Dana Scott and others. Is it then true that all the issues regarding recursive programs have now been settled in a satisfactory manner? Is it just a “classical” topic to which we can add nothing?

I think that it is not the case, in the sense that classical theories did not pay sufficient attention to how recursion is created, nor to how recursion interacts with other features of programming languages. I regard them as important and challenging questions, from both theoretical and practical perspectives. In fact, recent investigations on these issues have revealed quite rich — and abstract [9], I would say — mathematical structures behind recursive programs, which also have provided (or at least suggested) general — sometimes very concrete — principles for reasoning about recursion. I shall outline three such cases:

1. **Recursion from Cyclic Sharing** (in call-by-name languages) [3, 4, 5, 6] which explains recursion created from cyclic structures (like cyclic graph rewriting) using mathematics for cyclic structures (like knot theory) [10],

2. **Recursion and Control** (in call-by-value languages) [7, 8] which explains principles on recursion under the presence of side effects, especially control primitives like first-class continuations [2], and

3. **Recursion and Duality** (between call-by-name and call-by-value) [11, 12] which suggests a way to relate the seemingly isolated two cases explained above in terms of a mathematical duality [1, 13].

Through these stories, I shall try to advocate some views on the semantics-based approaches to programming languages which are now being carried out. In other words: “Programming Languages in the Abstract”.

Part of this talk is based on joint work with Yoshihiko Kakutani.
References


†: available from

http://www.kurims.kyoto-u.ac.jp/~hassei/papers/