Electronic Notes in Theoretical Computer Science 57 (2001)
URL: `http://www.elsevier.nl/locate/entcs/volume57.html` 4 pages

Hot Topics in Reduction Strategies - a panelist's view -

Tetsuo Ida [1] [,2]

Institute of Information Sciences and Electronics
University of Tsukuba, Tsukuba 305-8573, Japan

1 What are hot topics?

Let us first clarify the meaning of *hot* topics. There are at least two ways of explaining the *temperature* of the topics. The first one is the following. Since our research community is small, if several of us, who are inspired and guided some interesting previous results, are working on the same subject, that subject becomes noticeable in the community and becomes hot. The second is to attribute the temperature to social context. The research topic is hot if the research addresses the problems that our society is facing. The former is called *hot-by-community*, and the latter *hot-by-challenge* in this note. There will be combination of the two. Most researchers think that their research topics are *warm* in the measures of by-community and by-challenge.

I argue that the ideal situation is that our research topics are hot-by-challenge. However, in this world of advanced science and technology it is not easy to be in the ideal situation. One cannot easily find a problem that has a direct impact to our society. We would need a long chain of arguments even to justify our research. We tackle a problem P_0 since P_0 solves P_1, $\cdots$, and P_{n-1} solves P_n. The solution P_n is *tangible* to people. As n becomes larger, it becomes more difficult to convince people that P_0 is hot-by-challenge.

2 What is the reduction strategy?

I try to answer, in a manner of talking in a classroom of first-year computer science students, the above question without using rigorous formalism. This will be helpful to think about the relevance of the research on reduction strategies in wider context.

Suppose you are writing a program in a futuristic language, say a language similar to our natural languages in which you do not have to write much control

[1] This note is prepared for the panel on *Hot Topics in Reduction Strategies*.
[2] Email: `ida@score.is.tsukuba.ac.jp`

information of how to evaluate programs. You are not so much concerned with the evaluation order of program fragments (although of course you have to be conscious about evaluation order in some fragments of the program). Your program can be executed by a smart interpreter. You may find the interpreter not smart enough or not obedient enough to run your program that you wish. Then you look at your program and start to specify how the program should be evaluated.

You will have a variety of freedom of how to evaluate the program. You would like to have a guideline for the evaluation. This guideline is called *strategy.* The strategy may be to evaluate the program from left-to-right, right-to-left or something more complicated. When our program is a term to be rewritten by a set of term rewrite rules, the strategy is called *reduction strategy.*

You may then want to know, for example, if the strategies will deliver the same result for the same program. To answer the question, you would have to develop theories of reduction strategies.

3 What are hot topics in reduction strategies?

Let us now resume thinking about hot topics of reduction strategies. Reduction strategies are interesting since we study evaluation of programs. Programs are what fulfill our requests to computers. I am afraid that this argument is not exciting to many people, since it is too vague and plain. However, if we immediately start to talk about the definitions of reduction, reduction strategy, etc. in our formal language, people may not follow us.

For hot-by-challenge researches we need to find good applications, something in between computers and term rewriting, that appeal to people, figuratively speaking.

Below I will try to answer the questions posed by Gramlich[2].

4 What are the main challenges in research on reduction strategies?

I provide some research results and topics for the future that may answer the above question.

Theories of reduction strategies have been used successfully to explains the process of computation, more specifically the behavior of programs. Take for example the seminal work of Huet and Lévy on call-by-need computation[4]. It explains the essence of what is known the lazy evaluation of functional programming languages in very elegant and rigorous way. Their work not only explains the lazy evaluation, but clarifies the class of programs for which the soundness of lazy evaluation is assured.

Theories of reduction strategies have been used to design new programming languages. Lazy narrowing in functional logic programming is developed by lifting the concept of call-by-need computation to narrowing and then used to design our function logic programming language CFLP. Similar explanation may be possible for a function logic language Curry. For the computation models of those languages, see [5] and [3], respectively. The language ELAN[1] is more ambitious in exploiting the notion of reduction strategies. It can program reduction strategies.

As a concrete example of application of theories of reduction strategies, I would like to point out a symbolic computation language Mathmatica that are based on higher-order rewriting. Mathematica provides many useful functionalities with which we can design our own strategies. However, it seems that good theories that explain computation models are still missing.

Theories of reduction strategies can be used in program transformation. Program transformation is used to derive efficient programs from less efficient ones. One method to achieve this is to change evaluation orders without affecting the result of computation. Theories of reduction strategies can be used as a guideline to achieve this.

Many programs nowadays are running in distributed or parallel environment. Although theories for parallel and distributed computation are developed in different research communities, theories of reduction strategies may be able to provide a good theoretical basis for developing more sophisticated computing mechanisms.

5 Where is the significant potential for making relevant progress?

This question is difficult to answer. The question is more relevant when it addresses bigger problems involving more technologies, such as how to achieve truly ubiquitous computing.

I should hastily add, however, that term rewriting is an abstract notion of computing, and that if we hit the right point of complex computing systems, it is a powerful tool to analyze and understand them and moreover to present new models of computing systems in the relevant fields.

6 Where are promising links across different fields?

I already hinted the answer to this question. Reduction strategies are primarily related to programming. In the state of the art, reduction strategies are formulated within the framework of term rewriting. Therefore we can naturally link our researches on the reduction strategies to researches of functional and logic programs.

When we widen our objects of study from terms to those that model things that we daily handle, such as images, music and large textual documents, we will be able to locate more links with other fields. Although researches on rewriting on these objects are still cold-by-community, they are surely hot-by-challenge.

References

[1] P. Borovansky, C. Kirchner, H. Kirchner. A Functional View of Rewriting and Strategies for a Semantics of ELAN. In *Proc. of Fuji International Symposium on Functional and Logic Programming*, pages 143–166, Kyoto, 1998.

[2] B. Gramlich. Personal e-mail communication, May 13, 2001.

[3] M. Hanus. A Unified Model for Functional and Logic Programming. In *Proc. of the 24th Annual SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 80–93, Paris, 1997.

[4] G. Huet, and J. -J. Lévy. Computations in Orthogonal Rewriting Systems, I. In *Computational Logic: Essays in Honor of Alan Robinson*, pages 395–414, MIT Press, 1991.

[5] A. Middeldorp, S. Okui, and T. Ida. Lazy Narrowing: Strong Completeness and Eager Variable Elimination. *Theoretical Computer Science*, 167(1,2):95–130, 1996.