Program Transformation in Functional Programs

Yasunori Murashima

School of Information Science,
Japan Advanced Institute of Science and Technology

February 15, 1999

Keywords: program transformation, functional program, deforestation.

Program transformation aims at a generation of a more efficient program than an original one with preserving semantics. In program transformation, two properties are important. One is correctness that is defined by preserving semantics and improving efficiency. The other is termination. If program transformation does not terminate, it does not make sense. Program transformation contributes not only to increase the productivity of software but also to develop new algorithms.

In functional programming, program is made by composing small and basic functions. Most of basic functions are defined on such data structures as tree or list. Composing functions produces many intermediate data structures. These intermediate data structures cause loss of efficiency. In order to eliminate them, Wadler (1990) proposed deforestation.

Many researches on deforestation are discussed on lazy evaluation languages. However, deforestation on eager evaluation language has not been fully studied since some transformation rules destroy correctness.

Saga (1998) proposed partial evaluative deforestation that guaranteed termination of the transformation procedure and equivalence of transformed programs. Partial evaluative deforestation works on eager evaluation languages. His result shows that deforestation is effective in eager evaluation languages, though partial evaluative evaluation is not considered effectiveness.

In most results on deforestation, whether on eager or lazy evaluation languages, the occurrences of variables are too severe; the occurrence of the same variables in right-hand side of function definition is at most one. It is an obstacle to flexibility of programming.

In this paper, we propose blazed deforestation for eager evaluation that guarantees termination and correctness of transformations. Our procedure is designed based on Wadler's blazed deforestation.

Wadler's blazed deforestation (WBD) has the following merit.

- WBD allows existence of terms as intermediate data that does not cause inconvenience such as integer or Boolean values; It also allows variables that stands for such terms to exist more than once in the right-hand side of a function definition. Therefore, it can make programs more flexibly.

WBD has a merit compared to partial evaluative deforestation.

- A language that partial evaluative deforestation deals with needs to provide primitive functions such as "+" and "∗" as recursively defined programs. On the other hand, a language that WBD deals with can take them as library functions. Therefore, WBD can works on more realistic programming languages than partial evaluative deforestation does.

WBD has merits explained above, but it cannot be applied to programs on eager evaluation languages, because there exist some rules that do not hold correctness. Furthermore, WBD restricts on the occurrence of variables standing for terms such as tree and lists.

To solve these problems, in this study, we specify the rules that do not guarantee correctness in eager evaluation. It becomes clear that correctness does not hold for unfold and substitution. We show that correctness is regained by doing next two operations in such the cases.

- Add a condition in order to know whether the transformation rule is applicable. In lazy evaluation, there need not to give such conditions to ensure correctness, but in eager evaluation, the condition is indispensable.
- After transformation steps, a new operation to introduce a new function is applied. Correctness holds by doing this operation.

To remove the linearity restriction, we propose a new method based on that proposed by Wadler for WBD. When unfolding rule is applied, We use a Let expression to avoid repeated computations before unfolding. Furthermore, taking advantage of the merit of the eager evaluation strategy, we propose the method that unifies the occurrences of the same term into one occurrence so that the term is evaluated only once. Deforestation is a method that eliminates intermediate data structures, but in strict evaluation strategy, it appears that correctness holds by making use of intermediate data structures actively. The method unifying the same terms are done unless correctness is destroyed.

In this study, termination and correctness of the procedure of blazed deforestation for eager evaluation are proved. However, a general rate of efficiency cannot be discussed, because efficiency depends on the size of input data.

As described above, blazed deforestation for eager evaluation is proposed, and its effectiveness, problems and solutions are discussed. This study is hopeful that it is effective in expanding deforestation for eager evaluation, such as study on higher-order functions.