TRS compiler based on sequential strategy

Yasuo Seki

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

February 14, 1997

Keywords: term rewriting systems, functional language, compiler, normalizing strategy.

Recently, practical systems, Elan, Sml-NJ, Gofer, OBJ and so on, of functional languages have been developed. These systems have special features, i.e., delaied evaluation, higher-order functions, polymorphic type, etc. Other than these systems, there is a simple system, Cdimple, which generates programs written in procedural language C that execute TRS efficiently. Programs generated by Cdimple reduce terms given by user, in innermost strategy and outputs terms (normal form) having no reduces. Cdimple allows user defined C program together with input TRS. If is not garanteed that it caluculates the normal form even if the input term has a normal form, because of innermost strategy. This paper tries efficient implementation of Term Rewriting Systems (TRS) by improving Cdimple based on normalizing strategy.

In a class called Orthogonal, a term having normal form can be reduced to normal form by rewriting the position called index. G. Huet and J.-J. Le'vy have introduced a class of strong sequentiality (SS) that is subclass of orthogonal. They also show an automaton for searching for index of a given term. Althorgh an index of a given term t can be found efficiently by this artomaton, we always have to go back to root position of t after reductions. It is inefficient in case that repeatedly reducation are needed. Tus

Strandh propose Forward-Branching (FB) that is a sub class of SS. In this class, we can continue searching index without going back to the root position.

In this paper, we construct a TRS compiler that generate C program by unfolding FB index automaton into C procedures. Terms are represented as dag structure. Hence, redices are not duplicated by copying rules. Gabage collection is implemented by using reference counter. Since garbage collections are taken plase in real time, there is a little problem of overhead in operating on reference counters.

We compare our TRS-compiler with other systems (Cdimple, Elan compiler, Sml-NJ compiler, Gofer compiler).(See figure)

Cdimple, Elan compiler and Sml-NJ compiler adapt innermost reduction strategy. Gofer is based on leftmost outermost strategy. For some TRS's (factorial, fibonatch, naive reverse), our compiler generates several times slower code than systems based on innermost strategy. Our compiler produce a little faster program than Gofer.

For quick sort, our compiler takes much faster than Cdimple and Elan, Sml-NJ. However, ours is almost program that same speed as Gofer's. for the last TRS, the programs generated by our compiler get the answer, while others, are not terminate.

Table 1: results

	Our Compiler	Cdimple	Elan	SML-NJ	Gofer
fact 8	586ms	456ms	74ms	290ms	684ms
fib 20	509ms	478ms	113ms	120ms	722ms
naive reverse 1000	7712ms	9760ms	566ms	1230ms	9132ms
quiq sort 20	[a] 0ms	14086ms	2912ms	1850ms	2 ms
quiq sort 1000	543ms	[b] —	[c] △	[c] △	518ms
fbexamp	[a] 0ms	[d] ∞	[d] ∞	[d] ∞	[d] ∞

[a] $0 < 1$ms

[b] memory error

[c] long time $< \infty$, terminate

[d] non terminate