7R-06

Lazy Evaluation Schemes for Efficient Implementation of Multi-Context Algebraic Reasoning Systems

ChengCheng Ji[1,a)] Haruhiko Sato[1,b)] Masahito Kurihara[1,c)]

1. Introduction

An algebraic inductive theorem is a proposition for algebraic specifications defined on inductively-defined data structures such as natural numbers and lists. The proof of such inductive theorems plays a fundamental role in the field of formal verification of information systems. There are many single procedures of automatable proving methods such as KB [1] and RI [3]. Commonly, the nondeterministic choices of these procedures are essential to their successes. One may notice that running parallel processes could be a way to handle this problem. However, it is not a trivial task to implement these procedures in a physically paralleled way. Multi-context algebraic reasoning systems can efficiently simulate parallel processes each executing an algebraic reasoning procedure under a particular context (or a premise). Those systems are used to reason about algebraic computational systems such as term rewriting systems (TRSs), which are a concise and rigorous representation of computational systems in terms of rewrite rules. In fact, TRSs are studied and used in various areas of computer science, including automated algebraic inductive theorem proving, analysis and implementation of abstract data types, and decidability of word problems.

2. Multi-context Reasoning Systems

2.1 MKB

Knuth and Bendix have proposed a standard completion procedure called KB to generate a complete TRS [1]. Given a set of equations and a reduction ordering on a set of terms, the procedure uses the ordering to orient equations (either from left to right or from right to left to transform them into rewrite rules) and tries to generate a complete TRS equationally equivalent to the input set of equations.

However, the KB leads to three possible results: success, failure, or divergence. According to the possibility of divergence, we cannot try candidate orderings one by one. Also, it is not efficient to simply create processes for each different ordering and run them in parallel on a machine, because the number of candidate orderings normally exceeds ten thousands even for a small problem.

In 1999, this problem was partially solved by a completion procedure called MKB [2]. MKB is a single procedure that efficiently simulates execution of multiple processes each running KB with a different reduction ordering. The key idea of MKB lies in a data structure called node. The node contains a pair $s : t$ of terms and three sets of indices to orderings to show whether or not each process contains rules $s \rightarrow t$, $t \rightarrow s$, or an equation $s = t$. The well-designed inference rules of MKB allows an efficient simulation of multiple inferences in several processes all in a single operation.

DELETE: $N \cup \{\langle s : s, \emptyset, \emptyset, E\rangle\} \vdash N$
if $E \neq \emptyset$

ORIENT: $N \cup \{\langle s : t, R_0, R_1, E \cup E'\rangle\} \vdash$
$N \cup \{\langle s : t, R_0 \cup E', R_1, E\rangle\}$
if $E' \neq \emptyset, E \cap E' = \emptyset$,
and $s \succ_i t$ for all $i \in E'$

REWRITE_1: $N \cup \{\langle s : t, R_0, R_1, E\rangle\} \vdash$
$$N \cup \left\{\begin{array}{l}\langle s : t, R_0 \backslash R, R_1, E \backslash R\rangle \\ \langle s : u, R_0 \cap R, \emptyset, E \cap R\rangle\end{array}\right\}$$
if $\langle l : r, R, \ldots, \ldots\rangle \in N,\ t \rightarrow_{\{l \rightarrow r\}} u$,
$t \doteq l$, and $(R_0 \cup E) \cap R \neq \emptyset$

REWRITE_2: $N \cup \{\langle s : t, R_0, R_1, E\rangle\} \vdash N \cup$
$$\left\{\begin{array}{l}\langle s : t, R_0 \backslash R, R_1 \backslash R, E \backslash R\rangle \\ \langle s : u, R_0 \cap R, \emptyset, (R_1 \cup E) \cap R\rangle\end{array}\right\}$$
if $\langle l : r, R, \ldots, \ldots\rangle \in N,\ t \rightarrow_{\{l \rightarrow r\}} u$,
$t \rhd l$, and $(R_0 \cup R_1 \cup E) \cap R \neq \emptyset$

DEDUCE: $N \vdash N \cup \{\langle s : t, \emptyset, \emptyset, R \cap R'\rangle\}$
if $\langle l : r, R, \ldots, \ldots\rangle \in N$,
$\langle l' : r', R', \ldots, \ldots\rangle \in N, R \cap R' \neq \emptyset$,

[1] Division of Computer Science and Information Technology in Graduate School of Information Science and Technology, Hokkaido University, Sappro, 060-0814, Japan
[a)] kisyousei@complex.ist.hokudai.ac.jp
[b)] haru@complex.ist.hokudai.ac.jp
[c)] kurihara@ist.hokudai.ac.jp

and $s \leftarrow_{\{l \to r\}} u \to_{\{l' \to r'\}} t$

GC: $N \cup \{\langle s : t, \emptyset, \emptyset, \emptyset \rangle\} \vdash N$

SUBSUME: $N \cup \left\{ \begin{array}{l} \langle s : t, R_0, R_1, E \rangle \\ \langle s' : t', R_0', R_1', E' \rangle \end{array} \right\} \vdash$
$N \cup \{\langle s : t, R_0 \cup R_0', R_1 \cup R_1', E'' \rangle\}$
if $s : t$ and $s' : t'$ are variants and
$E'' = (E \backslash (R_0' \cup R_1')) \cup (E' \backslash (R_0 \cup R_1))$

2.2 MRIt

Term rewriting induction (RI) proposed by Reddy [3], is a automatable proof principle for proving inductive theorems on term rewriting systems. The RI method relies on the termination of the given term rewriting systems representing the axioms, because if we have a terminating term rewriting system (i.e., there exists no infinite rewrite sequence), we can use the transitive closure of the corresponding rewrite relation of the system as a wellfounded order over terms for the basis of induction.

However, there are several strategic issues in RI: (1) which reduction order should be applied, (2) which (axiomatic or hypothetical) rules should be applied during rewriting, and (3) which variables should be instantiated for induction.

The multi-context rewriting induction with termination checker (MRIt) solved these problems by creating virtual parallel processes dynamically to handle the nondeterministic choices.

3. Implementations and Experiments

We implemented both lz-mkb (based on MKB) and lz-itp (based on MRIt) by using lazy evaluation mechanism of the programming language Scala. The program was designed in an object-oriented way so that we could build and reuse the classes to organize the term structures, subsitutions, nodes, inference rules, etc. At the same time, we also followed the discipline of functional programming (e.g., “uniform return type” principle [4]) in coding so that it could be safer and easier to execute the program in a physically parallel computational environment.

To the lz-mkb case, we implemented the node (a basic unit of MKB) as as a class which contains an equation object as a datum and three bitsets as labels. We chose *bitset*[*1]to gain efficiency because there were numerous set operations during the computation. We also created a class called nodes for the set N of nodes for which several inference rules of MKB are defined. We embedded *lazy* values into these classes to gain efficiency. For example, during the whole procedure, the inference rule subsume would be invoked frequently due to the node comparisons. We created a *lazy* hash map $[\mathfrak{I}_s, \mathfrak{N}]$, where $\mathfrak{N}$ is a *list* of nodes and $\mathfrak{I}_s$ is a lazy value defined in the node class as the *size* of the node (i.e., for a node $n = \langle s : t, r_0, r_1, e \rangle$, $n.\text{size} = s.\text{size} + t.\text{size}$), so that we only need check the nodes with the same size as the original nodes. This check can be done efficiently by using the hash map with the node size as its key. In other words, for every $n \in N$, n uses its size $\mathfrak{I}_n$ as the key to $[\mathfrak{I}_s, \mathfrak{N}]$, then the set $\mathfrak{N}_n$ containing all the nodes with same size $\mathfrak{I}_n$ is looked up for the nodes with variant data. In our Scala program, the hash map $[\mathfrak{I}_s, N]$ is declared to be *lazy*, because it is calculated only once and then be stored as a constant object ready to be returned for repeated calculation requests afterwards.

[*1] a data structure defined in Scala's library

Table 1 computation time of mkb and lz-mkb

problem	mkb(ms)	lz-mkb(ms)	reduced time	reduced(%)
1	15003	1959	13044	86.94
2	160	130	30	18.75
5	14997	2738	12259	81.74
8	275	205	70	25.45
11	90	60	30	33.33
14	480	351	129	26.88
17	85	65	20	23.53
19	730	471	259	35.48
30	140	95	45	32.14
avg.	-	-	-	40.47

The computation time for each examined completion problems is summarized in Table 1. The results obtained by the program using the lazy evaluation are labeled lz-mkb, and those obtained by the original one are labeled mkb. We can see, lz-mkb is more efficient (about 40% faster in average) than mkb in all the problems examined.

Since lz-itp also works on the set of nodes, we exploited the lazy evaluation scheme for the nodes manipulation to gain more efficiency. Moreover, we implemented the lemma exploration function with divergence analyzation [5] to deal with the lemma-required problems. From the Table 2, we can see lz-itp which used the lazy evaluation schemas was more efficient than mrit.

Table 2 Computation time of mrit and lz-itp

problem	lem_1	lem_2	lem_3	lem_4	lem_5	lem_6
lz-itp	602	932	12379	17738	1050	1023
mrit	615	969	12801	18090	1075	1049

4. Summary

We have presented two new implementations of the multi-context reasoning system: lz-mkb and lz-itp. We applied lazy evaluation schemas with several heuristic ideas in our implementation [6] [7]. The experiments show that our new implementations are more efficient than the original ones in all the problems examined.

References

[1] D. E. Knuth and P. B. Bendix, “Simple word problems in universal algebras,” *J. Leech(ed.), Computational Problems in Abstract Algebra*, Pergamon Press, 1970, pp.263-297.

[2] M. Kurihara and H. Kondo, “Completion for multiple reduction orderings,” *Journal of Automated Reasoning*, Vol.23, No.1, 1999, pp.25-42.

[3] U. Reddy, “Term rewriting induction,” *10th Int. Conf. on Automated Deduction, vol.814 of Lecture Notes in Computer Science*, pp.162-177, 1990.

[4] M. Odersky, *Programming in Scala,* 2nd ed., Artima Press, 2010.

[5] T. Walsh , “A divergence critic for inductive proof”, *Journal of Artificial Intelligence Research*, vol. 4, pp. 209-235, 1996.

[6] CC. Ji, H. Sato, M. Kurihara, "Lazy Evaluation Schemes for Efficient Implementation of Multi-Context Algebraic Completion System," *IAENG International Journal of Computer Science*, vol. 42, no. 3, pp.282-287, 2015.

[7] CC Ji, H. Sato, and M. Kurihara, "A New Implementation of Multi-Context Algebraic Inductive Theorem Prover," *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering and Computer Science 2015*, 21-23 October, 2015, San Francisco, USA, pp.109-114.