Improving Performance of Automated Forward Deduction System EnCal on Shared-Memory Parallel Computers

Yuichi Goto
Graduate School of Science and Engineering,
Saitama University
255 Shimo-Okubo, Saitama-shi,
Saitama 338-8570, Japan
gotoh@aise.ics.saitama-u.ac.jp

Daisuke Takahashi
Institute of Information Sciences and Electronics,
University of Tsukuba
1-1-1 Tennodai, Tsukuba-shi,
Ibaraki 305-8573, Japan
daisuke@is.tsukuba.ac.jp

Jingde Cheng
Graduate School of Science and Engineering,
Saitama University
255 Shimo-Okubo, Saitama-shi,
Saitama 338-8570, Japan
cheng@ics.saitama-u.ac.jp

Abstract

The performance of an automated forward deduction system is crucial to its applicability. Since a forward deduction system working for discovery has no explicitly specified proposition or theorem given previously as goal, it often deduces many redundant intermediates, i.e., instances of those that have previously deduced. Therefore, how to reduce redundant intermediates is a general and difficult issue for any forward deduction system. In particular, since a parallel forward deduction system has multiple threads, the issue is more crucial to its performance. In this paper, we present a new algorithm to detect redundant intermediates in order to improve performance of EnCal, an automated forward deduction system for general-purpose entailment calculus. We have implemented EnCal based on the new algorithm on a shared-memory parallel computer and our experiment showed this algorithm is effective.

1. Introduction

An automated forward deduction system for entailment calculus is an indispensable component of many application systems such as theorem finding and knowledge discovery systems that require a forward reasoning engine. The performance of an automated forward deduction system is crucial to its applicability. Since a forward deduction system working for discovery has no explicitly specified proposition or theorem given previously as goal, it often deduces many redundant intermediates, i.e., instances of those that have previously deduced. Therefore, how to reduce redundant intermediates is a general and difficult issue for any forward deduction system. In particular, since a parallel forward deduction system has multiple threads, the issue is more crucial to its performance.

On the other hand, reasoning rule generation and automated theorem finding [1, 9, 10] are two fundamental issues in knowledge engineering. To solve these two problems, it is essential to establish a domain-independent fundamental theory underlying an autonomous reasoning mechanism, and then to develop automatic reasoning tools based on that mechanism. Cheng proposed some strong relevant logics and showed that an entailment calculus based on those logics can underlie reasoning rule generation in knowledge-based systems and automated theorem finding [1, 2, 4]. We are developing an automated forward deduction system for general-purpose entailment calculus, named EnCal, which supports entailment calculus based on strong relevant logics as well as other logics [3]. However, there are two problems for that EnCal serves as a practical automated reasoning engine: a large amount of execution time and a large amount of memory needed for generating a large number of logical theorems [6]. In order to solve those problems, we have implemented a parallelization version of EnCal on a shared-memory parallel computer. Although we successfully achieved speed-up ratio of about 12 times on 16 processor's shared-memory parallel computer [5], we faced the problem that the number of redundant intermediates increases in proportion to the increase of the number of processors. Since the problem exacerbates the performance of EnCal, it is necessary to reduce redundant intermediates.

In this paper, we propose a new algorithm to detect and reduce redundant intermediates in order to improve performance of parallelization version of EnCal. We have implemented EnCal based on the new algorithm on a shared-memory parallel computer and our experiment showed this algorithm is effective.

2. Forward deductions by EnCal

2.1. EnCal and its processing features

IF-THEN rules have played and they are still playing various important roles in knowledge-based systems. In logic, a sentence in the form of "if . . . then . . ." is usually called *conditional* or *entailment*. The notion of entailment plays the most essential role in reasoning because any reasoning form must invoke it. *Entailment calculus* is a method to formalize a logical system L where the notion of entailment is represented by a primitive connective such that its logical theorems are represented in the form of entailment. *A forward entailment calculus* is an entailment calculus to deduce new theorems by applying inference rules of L to given axioms of L. EnCal supports forward entailment calculus based on strong relevant logics as well as other logics. It provides its users with the following major facilities [3]. For a logic L which may be a propositional logic, a first-order predicate logic, or a second-order predicate logic, a non-empty set P of formulas as premises, inference rules of logic system L and natural number k and j as limit of degree which is the degree of nested entailment (denoted by "$\Rightarrow$" in this paper), all specified by the user, EnCal can (1) reason out all logical theorem schemata of the k^{th} degree fragment of L, (2) verify whether or not a formula is a logical theorem of the k^{th} degree fragment of L, if yes, then give the proof, (3) reason out all empirical theorems of the j^{th} degree fragment of the formula is an empirical theorem of theory with premises P , and (4) verify whether or not a formula is an empirical theorem of theory with premises P, if yes, then give the proof [3].

EnCal consists of six parts as follows.

1. Initialization part: it takes in premises, limit of degree k and j and inference rules.

2. Forward deduction part: it repeats following process until it deduces no new theorems.

   (a) Matching part: it searches and picks up a previously deduced theorem to apply an inference rule.

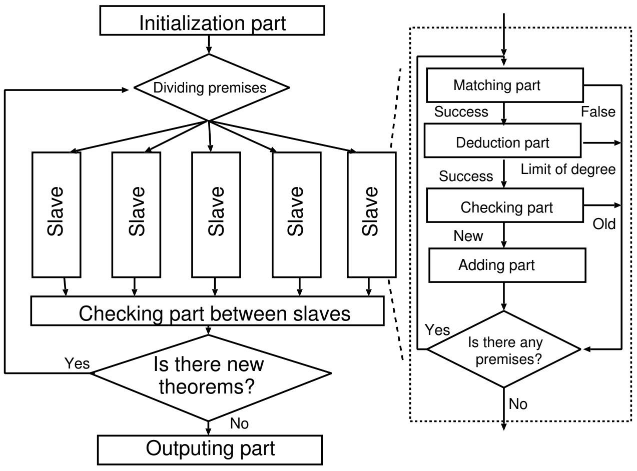

Figure 1. The model of parallelization version of EnCal.

   (b) Deduction part: it applies the inference rule to the picked up theorem at "Matching part". At present, the inference rule is Modus Ponens only. Modus Ponens is that "B" is deduced from "$A \Rightarrow B$" and "A".

   (c) Checking part: it checks all previously deduced theorems and a deduced theorem at "Deduction part" in order to check whether the deduced theorem is new or redundant.

   (d) Adding part: it adds a new theorem to premises if the degree of new theorem is n^{th} degree ($1 \leq n \leq k$).

3. Outputting part: it outputs deduced new theorems to file.

EnCal takes a large time when it deduces a large amount of theorem schemata. For example, when the number of deduced theorem schemata is 3,595,264, it took 522 hours on Sun Enterprise 6000. The great portion of the execution time is spent at "Checking part". Since an theorem schema substitute pattern variable of other theorem schema, the number of form of the theorem schema is infinite. Symbolic manipulation to transform a theorem schema to a condensed form, and to compare deduced theorem with previously deduced theorems is needed to detect redundant theorem. *condensed form* means a form whose other abbreviated form does not exist. We call this symbolic manipulation "detection of redundant theorem schemata", DR for short. DR is very high-frequency process in EnCal. The frequency of DR has a great influence on the execution time of EnCal.

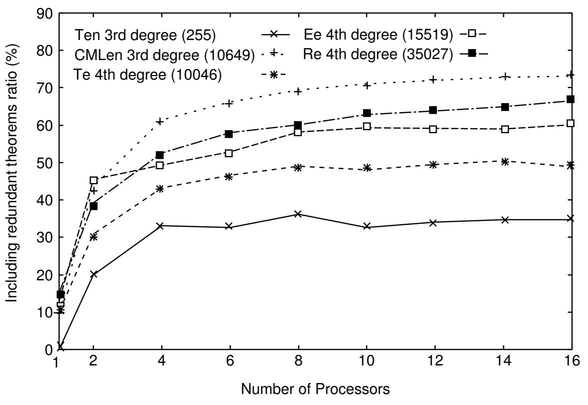

Figure 2. Including the redundant theorems ratio.

2.2. A parallelization version of EnCal

We have implemented a parallelization version of EnCal based on the master-slave model on a shared-memory parallel computer in order to improving performance of EnCal [5]. This model is shown in Fig. 1.

1. Initialization part: it takes in premises, limit of degree k and j and inference rules.
2. Master part: This part changes "Slave part" after dividing premises to "Slave parts".
3. Slave part: The following processing are repeated independently of other "Slave part".
   (a) Matching part: it searches and picks up a previously deduced theorem to apply an inference rule.
   (b) Deduction part: it applies the inference rule to the picked up theorem at "Matching part". At present, the inference rule is Modus Ponens only.
   (c) Checking part: it checks all previously deduced theorems and the deduced theorem at "Deduction part" in order to check whether deduced theorem is new or redundant.
   (d) Adding part: it adds a new theorem to premises if the degree of new theorem is n^{th} degree ($1 \leq n \leq k$).
4. Checking between slaves part: it checks deduced theorems which are not reduced "Checking part" at "Slave part".
5. Outputting part: it outputs deduced new theorems to file.

It is necessary for efficient detection of redundant theorems to be able to access all previously deduced theorems. However, the set of deduced intermediates at a "Slave part" is not referred by other "Slave parts". The parallelization version of EnCal needs "Checking part between slaves".

In our experiment, the parallelization version of EnCal achieved speed-up ratio of about 12 times on 16 processor's shared-memory parallel computer. However, we faced a problem that the number of redundant theorems increases in proportion to the increase of the number of processors. Fig. 2 shows that the including redundant theorems ratio depends on the number of processors. "Including redundant theorems ratio" is the number of redundant theorems in finally deduced theorems is divided by the number of finally deduced theorems. If "Including redundant theorems ratio" is 0 percent, then there are no redundant intermediates in finally deduced theorems. In Fig. 2, "Ten 3rd degree (252)" means that 252 core theorems ("core theorems" means that finally deduced theorems do not include redundant theorems) are deduced from axioms of the relevant logic system T with entailment and negation. "CMLen 3rd degree (10649)" means that 10649 core theorems are deduced from axioms of classical mathematical logic with entailment and negation. "Te 4th degree (10046)" means that 10046 core theorems are deduced from the axioms of relevant logic system T with entailment. "Ee 4th degree (15519)", "Re 4th degree (35027)" and so on.

3. A new algorithm to detect redundant intermediates

In order to solve the problem mentioned in Section 2, we investigate why the number of redundant theorems increase in proportion to the increase of the number of processors.

The reason is that the method to detect redundant theorems is inefficient in"Checking part" and "Checking part between slaves". By comparing the deduced theorem at "Deduction part" with previously deduced theorems,it judges whether the deduced theorem is a new theorem. The algorithm of "Checking part" is as follows. Here, REFER denotes the set of previously deduced theorems and TARGET denotes a deduced theorem at "Detection part".

```
$Checking(\text{REFER}, \text{TARGET})$
1   $R \leftarrow$ a theorem $\in$ REFER
2   while $R \neq$ NIL
3     do if $DR(R, \text{TARGET}) =$ "TARGET is redundant"
4       then return "TARGET is redundant"
5     else $R \leftarrow$ a theorem $\in$ REFER
6   return "TARGET is new"
```

$Checking(\text{REFER}, \text{TARGET})$ returns "TARGET is new" or "TARGET is redundant".

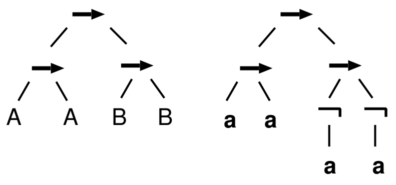

Figure 3. Represent by tree structure.

Table 1. Checking node matrix.

		kind of B nodes				
		$\Rightarrow$	$\wedge$	$\vee$	$\neg$	variable
kind of A nodes	$\Rightarrow$	α	β	β	β	γ
	$\wedge$	β	α	β	β	γ
	$\vee$	β	β	α	β	γ
	$\neg$	β	β	β	α	γ
	variable	δ	δ	δ	δ	ϵ

$DR(R,T)$ denotes a function to transform a theorem schemata to condensed form and to compare deduced theorem with a previously deduced theorem. We represent that $\Rightarrow$, $\wedge$ and $\vee$ are node of binary tree, $\neg$ is node of list, and argument of theorem is label of these tree. For example, "$(A \Rightarrow A) \Rightarrow (B \Rightarrow B)$" and "$(a \Rightarrow a) \Rightarrow (\neg a \Rightarrow \neg a)$" represent like Fig. 3. If there are theorem A and theorem B, $DR(R,T)$ returns "B is new" or "B is redundant". The algorithm of $DR(R,T)$ as follows. R is a pointer of node in theorem A. T is a pointer of node in theorem B.

```
DR(R,T)
1  if instance of R is a pattern variable.
2     then if bind(R,T) = SUCCESS
3        then return "B is redundant"
4     else return "B is new"
5  else if instance of R = instance of T
6     then R' ← a pointer of left node of R
7     T' ← a pointer of left node of T
8     if DR(R',T') = "B is new"
9        then return "B is new"
10    else R' ← a pointer of right node of R
11       T' ← a pointer of right node of T
12       if DR(R',T') = "B is new"
13          then return "B is new"
14       else return "B is redundant"
15    else return "B is new"
```

$bind(A,B)$ is function to judge consistency of mapping sub-theorem to a pattern variable. A is a pattern variable, B is a pointer of root in a sub-theorem. $bm[]$ is array that stored sub-theorem of which B is root. The index of $bm[]$ corresponds to the kind of pattern variable. The algorithm of $bind(A,B)$ as follows.

```
bind(A,B)
1  if bm[A] = NIL
2     then bm[A] ← a sub-theorem of which B is root
3     return SUCCESS
4  else if bm[A] = a sub-theorem of which B is root
5        then return SUCCESS
6     else return FAILURE
```

In "Checking part" and "Checking between slaves part", DR is applied only one time to previously deduced theorems each time a theorem is deduced at "Deduction part". There are A and A' which is redundant of A. If A was deduced earlier than A', A' is detected as a redundant theorem by DR. If A' is deduced earlier than A, A' is never detected by DR. In forward entailment calculus, it is not always that a theorem which is a condensed form is deduced earlier than redundant theorems. Hence, the processing in "Checking part" and "Checking between slaves part" has possibilities not to detect redundant theorems. In order to solve the problem not to detect redundant theorems, DR is applied only two time to previously deduced theorems each time a theorem is deduced at "Deduction part". Since the frequency of DR has a great influence on the execution time of EnCal, the increase of the frequency of DR is wrong with improving the performance of EnCal. Thus we propose a new algorithm of DR which focus on structure of entailment in order to solve the problem not to detect redundant theorems. The execution time of the implementation based on the new algorithm is shorter than the execution time to do old DR twice. A new algorithm of DR is as follows. R is a pointer of node in theorem A. T is a pointer of node in theorem B.

```
new-DR(R,T,state)
1  arg ← pair(R,T)
2  if arg = α
3     then R' ← a pointer of left node of R
4     T' ← a pointer of left node of T
5     if (state ← DR(R',T',state)) = "B is new"
6        then return "B is new"
7     else R' ← a pointer of right node of R
8        T' ← a pointer of right node of T
9        if (state ← DR(R',T',state)) = "B is new"
10          then return "B is new"
11       return state
12 elsif arg = β
13    return "B is new"
14 elsif arg = γ
15    then if state = "Both theorem are same"
16       then state ← "A is redundant"
17    elsif state = "B is redundant"
18       then return "B is new"
19    if bind(T,R) = FAILURE
```

Table 2. The number of redundant theorems in finally deduced theorems.

Number of processors	Number of redundant theorems			
	CMLen (3)		Re (4)	
	Old	New	Old	New
1 processors	1159	0	6253	0
2 processors	8227	0	22542	0
4 processors	16920	0	38899	0
8 processors	24250	0	52883	0
16 processors	28894	0	69434	0

```
      **then return** "$B$ is new"
    **else return** $state$
  **elsif** $arg = \delta$
    **then if** $state =$ "Both theorem are same"
      **then** $state \leftarrow$ "$B$ is redundant"
    **elsif** $state =$ "$A$ is redundant"
      **then return** "$B$ is new"
    **if** $bind(R,T) =$ FAILURE
      **then return** "$B$ is new"
    **else return** $state$
  **else if** $state \neq$ "$A$ is redundant"
      **then if** $bind(R,T) =$ FAILURE
        **then return** "$B$ is new"
    **if** $state \neq$ "$B$ is redundant"
      **then if** $bind(T,R) =$ FAILURE
        **then return** "$B$ is new"
      **return** $state$
```

$pair(R,T)$ is a function that gives back return value on the based on table 1. $state$ is a relation of two theorems. If there are theorem A and theorem B, then relation between theorem A and theorem B is one of the four states; both theorems are same, B is a redundant theorem of A, A is a redundant theorem of B. B is different from A. $DR(R,T,state)$ returns one of the four states. If a theorem which is a condensed form is deduced earlier than redundant theorems, new DR can detect those redundant theorems. Although the frequency of processing DR is same, the new DR can detect redundant theorems which old DR cannot detect.

4. Results and Discussion

We have implemented a parallelization version of EnCal based on the new algorithm with C and Open MP [7, 8] on the Sun Enterprise 6000 (16 processors). We also compared the performance of the parallelization version of EnCal based on the new algorithm, new implementation for short, with the old parallelization version of EnCal, old implementation for short. The old implementation has "Last Checking part" before "Outputting part". "Last Checking part" is to detect and reduce redundant theorems which are not reduced at "Checking part" and "Checking between slaves part".

Table 2 shows the number of redundant theorems in finally deduced theorems. In table 2, "CMLen (3)" denotes that premises are CMLen and limit of degree is 3rd, "Re (4)" denotes that premises are Re and limit of degree is 4th. "Old" means the old implementation. "New" means the new implementation. "Number of processors" means the number of used processors to deduced theorems. "Number of redundant theorems" means the number of redundant intermediates which are detected and reduced at "Last Checking part".

Table 3 shows the execution time of both parallelization versions of EnCal on 16 processors. "Execution time before Last Checking part" denotes the execution time to complete all deduction. "Execution time at Last Checking part" denotes the execution time of processing "Last Checking part". "Execution time (total)" denotes the execution time of processing from "Initialization part" to "Last Checking part". The execution time before "Last Checking part" shows that the execution time of the new implementation is about 1.1 times as slow as the old one. However, the execution time at "Last Checking part" of the new implementation is zero. It is not necessary for the new implementation to set "Last Checking part", because there are no redundant intermediates in finally deduced theorems. Thus total execution time of the new implementation is faster than the old one.

Our experimental result shows us three things. First, the new algorithm is effective to detect and reduce redundant theorems. Second, the execution time of new implementation is shorter than twice processing DR. Third, we can get the deduced theorems which are no including redundant theorems in short time than old one. Thus, without caring about the increase of the number of redundant theorems, we can get the high speed-up ratio by increase of the number of processors.

Table 3. The execution time (used 16 processors) [second].

		Premises (the number of degree)			
		CMLen (3)	Te (4)	Ee (4)	Re (4)
Execution time before "Last Checking part"	Old	1058	323	1155	7903
	New	1165	373	1271	8224
Execution time at "Last Checking part"	Old	317	488	1129	6062
	New	0	0	0	0
Execution time (total)	Old	1377	804	2273	13974
	New	1165	373	1271	8224

5. Concluding remarks

We have proposed a new algorithm to detect and reduce redundant intermediates in order to solve the problem that the number of redundant intermediates increases in proportion to the increase of the number of processors. We have implemented EnCal based on the new algorithm on a shared-memory parallel computer and our experiment showed this algorithm is effective. Therefore, we can get the high speed-up ratio, free from worry about the increase of the number of redundant intermediates, by increase of the number of processors.

A forward deduction system to deal with theorem schemata needs some specific method to detect redundant intermediates. Since a redundant intermediate is not always deduced earlier than a condensed form of the redundant intermediate, the method to detect redundant intermediate instances is not enough to detect redundant intermediate schemata. Our algorithm to detect redundant intermediate schemata is to focus on the structure of entailment formulas. Therefore, this algorithm is effective for not only improving performance of EnCal, but also other automated forward deduction system based on entailment calculus.

References

[1] J. Cheng. Entailment calculus as a logical tool for reasoning rule generation and verification. In J. Liebowitz, editor, *Moving Toward Expert Systems Globally in the 21st Century*, pages 386–392. Cognizant Communication Co., 1994.

[2] J. Cheng. Entailment calculus as the logical basis of automated theorem finding in scientific discovery. In V.-P. Raul, editor, *Systematic Methods of Scientific Discovery: Papers from the 1995 Spring Symposium*, pages 105–110. AAAI Press - American Association for Artificial Intelligenc, 1995.

[3] J. Cheng. Encal: An automated forward deduction system for general–purpose entailment calculus. In N. Terashima and E. Altman, editors, *Advanced IT Tools, Proceedings of the 14th WCC, Canberra*, pages 507–517. Chapman & Hall, 1996.

[4] J. Cheng. The fundamental role of entailment in knowledge representation and reasoning. *Journal of Computing and Information*, 2(1):853–873, 1996.

[5] Y. Goto, D. Takahashi, and J. Cheng. Parallel forward deduction algorithms of general-purpose entailment calculus on shared-memory parallel computers. In *Proc. ACIS 2nd International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel/Distributed Computing (SNPD01)*, pages 169–175, Nagoya, Japan, 2001.

[6] K. Nishi, J. Cheng, and K. Ushijima. Improving the performance of automated forward deduction system encal. In *Proc. International Symposium on High Performance Computing (ISHPC'97)*, volume 1336 of *Lecture Notes in Computer Science*, pages 371–380. ACM, Springer-Verlag, 1997.

[7] Omni. RWCP OpenMP compiler project. http://www.hpcc.jp/Omni/.

[8] OpenMP. Simple, Portable, Scalable SMP Programming. http://www.openmp.org/.

[9] L. Wos. *Automated Reasoning: 33 Basic Research Problems*. Prentice-Hall, 1988.

[10] L. Wos. The problem of automated theorem finding. *Journal of Automated Reasoning*, 10(1):137–138, 1993.