2012 IEEE International Conference on Systems, Man, and Cybernetics
October 14-17, 2012, COEX, Seoul, Korea

Concurrency Control Program Generation by Decreasing Nodes of Program Trees in Genetic Programming

Shinji Tamura Teruhisa Hochin Hiroki Nomiya
Graduate School of Science and Technology
Kyoto Institute of Technology
Kyoto, Japan
{hochin, nomiya}@kit.ac.jp

Abstract- **This paper tries to generate an appropriate concurrency control program by using genetic programming (GP). In GP, a program is represented with a tree. Nodes of a tree are selected from a symbol set. This paper tries two symbol sets: the high-level symbol set and the reduced one. The high-level one includes high-level symbols created by combining conventional ones. In the reduced symbol set, symbols are drastically decreased by changing the method of implementing the concurrency control program. Automatic defined functions (ADFs) are also used. Introducing high-level symbols caused the increase of the number of symbols. This made the program generation difficult. On the other hand, an appropriate program could be generated with the reduced symbol set. An ADF is also used in the program generated.**

Keywords-concurrency control; program generation; genetic programming; granurality

I. INTRODUCTION

Information and network technologies have rapidly and widely spread. Databases have been used in various application areas. Database management systems are used in these areas. It tends to be difficult that a monolithic database management system fits all of database application areas. The database management system that could be adapted to each database application area will be required [1, 2].

This paper focuses on the concurrency control mechanism, which is a very important component of the database management system because it guarantees the reliability and influences the performance of database processing [3]. A series of database operations could be treated as a unit in the database management system. This unit is called a transaction. Features of transactions differ according to the database application areas. Examples of the features are the number of operations, the ratio of write operations to read ones, the number of tables manipulated, and the time period of the first operation to the last one. These features influence the concurrent execution of database operations. When all of operations are read ones, all of operations could concurrently be executed. When all operations are the write one updating the value of the same data item of the same table, all of the operations must serially be executed. That is, no operations could concurrently be executed. The concurrency control mechanism controls concurrent execution of the operations. The concurrency control mechanism suited to a database application area may be different from that suited to another one. The database management system, however, supports only one concurrency control mechanism in spite of kinds of database application areas. Overhead of concurrency control, which may not be needed for a specific database application, may have to be paid. This may degrade the performance of database processing. The concurrency control mechanism suited to each database application is preferred.

We have proposed a generation method of concurrency control program suited to a specific database application by using Genetic Programming (GP) [10,11]. The functions and the terminals of program, and the fitness measure function used in GP have been proposed. The functions and the terminals include those for the management of concurrent execution of database operations. This management is captured as procedures using the variables attached to data items and transactions. The experiments of the program generation showed that the popular locking algorithm could be generated under the concurrent environment, while the algorithm better than the popular locking algorithm could be generated under the not-so-concurrent environment [10]. Moreover, the generation method has been extended to support semantic concurrency control [11].

Although some programs could be generated, the programs generated are similar to the conventional concurrency control programs. This may be due to the initial program specified. If the initial programs are not specified, the appropriate programs are not generated. This may be caused by a large number of functions and terminals of programs.

This paper tries to generate appropriate concurrency control programs by using GP. The usage of automatic defined functions (ADFs), the introduction of high-level functions and terminals, and the reduction of the functions and terminals are examined. It is experimentally shown that the reduction of functions and terminals is effective, while the others are not.

The remaining of this paper is as follows: Section II describes GP and concurrency control mechanism. The generation method of concurrency control mechanism is described in Section III. Section IV describes several approaches in order to generate appropriate concurrency control programs by using GP. The experiments of the program generation are conducted in Section V. Section VI gives some considerations. Section VII concludes this paper.

978-1-4673-1714-6/12/$31.00 ©2012 IEEE

II. PREPARATION

A. Genetic Programming

Genetic Programming (GP) is a biologically inspired method that is able to create computer programs from a high-level problem statement [8, 9]. GP works on populations of solution candidates for a given problem and is based on Darwinian principles of survival of the fittest (selection), recombination (crossover), and mutation. GP generates the program suited to the situation.

In GP, computer programs are represented with rooted, labeled structure trees, which are called program trees in this paper. Nodes of trees are functions or terminals. Terminals are evaluated directly, while functions are evaluated after the evaluation of children's nodes. Functions and terminals are called *symbols* in this paper. An example of a program is shown in Fig. 1. The leaf nodes of the program tree are terminals. The other nodes are functions. The root node, which is the function *IF*, evaluates the condition specified at the first argument, which is the left sub-tree. The left sub-tree represents the condition whether the variable *X* is larger than the variable *Y*. When the condition is satisfied, the second argument, which is the right sub-tree, is invoked. In the right sub-tree, the value of the variable *Z* is set to zero.

Before GP begins, a symbol set of program trees, a fitness measure function, termination criterion, and parameters are prepared. Based on them, the execution of GP begins. At the beginning, the population of programs is arbitrarily initialized. Then, the genetic programming cycle begins. The fitness of a program in the population is calculated by using the fitness measure function. Based on the fitness obtained, some programs are selected. By applying the genetic operations, i.e., mutation, crossover, and copy, to the programs selected, the next population of programs is created. When the termination condition, which is usually on the number of generations, meets, the genetic programming cycle terminates, and the best program is obtained.

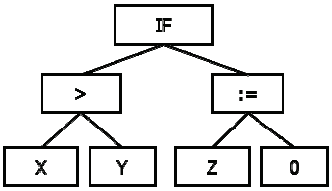

Figure 1. Example of a program used in GP.

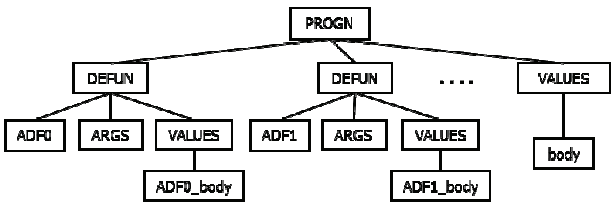

Figure 2. A program tree including ADF definitions.

The automatic defined function (ADF) is one of the well-known extensions to the basic program trees [9]. The main idea of ADFs is that program code is organized into useful groups (subroutines). This enables the parameterized reuse of code. The data structure of a program using ADFs is shown in Fig. 2. The program tree of the main program resides at the right sub-tree of the root node. Sub-trees residing at the left of the root node are for ADFs. The program of an ADF resides at the sub-tree under the node VALUES. The ADFs defined at the left of the root node could be used in the main program. For example, in the case of ADF0, the function named ADF0 can be used in the main program.

Please note that genetic operations are applied to the main programs, or the *i*th ADFs. That is, genetic operations are applied to neither the main program and an ADF, nor the *i*th ADF and *j*th ADF.

B. Concurrecy control

Transaction-processing systems usually allow multiple transactions to run concurrently [3]. Here, a transaction is a collection of operations that form a single logical unit of work. Allowing multiple transactions to update data concurrently causes several complications with consistency of the data. Ensuring consistency in spite of concurrent execution of transactions requires extra work. This work is called concurrency control. As the serial execution of transactions never violates consistency, it is the criterion of keeping consistency. The schedule which is obtained by putting the operations of transactions one transaction by one transaction is called a serial schedule. When the effect of a schedule is the same as that of a serial schedule, consistency is kept. Such a schedule is called a serializable schedule.

The most popular method is to allow a transaction to access a data item only if it is currently holding a lock on that item. There are usually two kinds of mode: a shared-mode lock, and an exclusive-mode lock. If a transaction *Ti* has a shared-mode lock on a data item *Q*, then *Ti* could read *Q*, but could not write *Q*. Another transaction *Tj* could also read *Q*. If a transaction *Ti* has an exclusive-mode lock on a data item *Q*, then *Ti* could read and write *Q*. No other transactions could read or write *Q*. A shared-mode (exclusive-mode, respectively) lock is also called a read (write) lock. One protocol ensuring consistency in spite of concurrent execution of transactions is the *two-phase locking protocol (2PL)*. This protocol requires that each transaction issues lock and unlock requests in two phases: the growing phase and the shrinking one. Once a transaction releases a lock, the phase of the transaction turns to the shrinking phase from the growing one, and the transaction could not acquire any new locks.

Another method is to select an ordering among transactions in advance. The most popular method is a *timestamp-ordering protocol (TSO)*. In this protocol, a timestamp is assigned to each transaction. The timestamp may be a value of the system clock of the computer. The timestamps of transactions determine the order of the execution.

These protocols guarantee that the schedules are serializable.

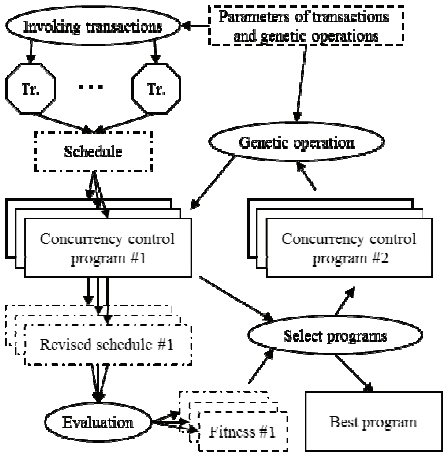

Figure 3. Outline of generation system of concurrency control program.

III. CONCURRENCY CONTROL PROGRAM GENERATION SYSTEM

A. Outline

A concurrency control program controls transactions to be concurrently executed with ensuring the consistency of data. The concurrency control program suited to an application program is tried to be generated by using GP [10,11]. The outline of the generation system is shown in Fig. 3. The steps of this system are as follows:

(1) Parameters for specification of transactions and genetic operations are specified.

(2) Transactions are invoked.

(3) A schedule is obtained by ordering the operations issued from the transactions according to the order of the generation time of the operations.

(4) Concurrency control programs generated revise a schedule.

(5) Fitness of the concurrency control program is obtained by evaluating the revised schedule.

(6) Concurrency control programs are selected from the current ones based on the fitness.

(7) Genetic operations are applied to the concurrency control programs selected in Step (6) to generate the next generation's programs.

(8) If the number of generations does not reach the limit, then repeat from Step (4). Otherwise, the generation system terminates, and the best concurrency control program is obtained.

The roulette-wheel selection is used as the selection method in Step (6). A selection is made with probability proportional to the fitness value. A roulette-wheel, whose sectors' widths are proportional to the fitness value, is spun, and the sector where the ball falls is selected [9].

Parameters specified in Step (1) are explained here after. These are the parameters of specifying transactions and genetic operations.

1) Specification of transactions: The parameters of the features of transactions are the number of transactions concurrently executed (p1), the number of data items manipulated by a transaction (p2), the ratio of write operations to all of operations (p3), the maximum number of operations a transaction (p4), and the probability of abortion of a transaction (p5). The larger p1 and p3 are, the more conflicts arise because many requests of writing data to the same data item occur. The less p2 is, the more conflicts arise because of the same reason described above.

The transactions invoked in Step (2) issue operations according to these parameter values.

2) Specification of genetic operations: The parameters of genetic operations are the maximum number of generations (s1), the number of programs a generation (s2), specification of the initial program (s3), the maximum depth of a program (s4), the growing method of a program (s5), the probability of crossover of functions (s6), the probability of crossover of terminals (s7), the probability of copying a program (s8), the number of schedules used in the learning (s9), the number of schedules used in the evaluation (s10), the preference of small programs (s11), specification of the initial ADF program (s12), the maximum depth of ADFs (s13), and the growing method of ADFs (s14).

When the existence of initial program is specified by s3, a program written in S-expression is required. The growing method specified by s5 is that the program randomly grows, or that it grows until the depth specified by s4. The possibility of mutation is calculated by the equation 1.0 – s6 – s7 – s8. The fitness calculation considers s9 and s10. The parameters s12 to s14 are for ADFs. These correspond to s3 to s5 for the initial program, respectively.

The function and terminal sets as well as a fitness measurement function have to be determined for using GP. These are explained as follows.

B. Structure of program

As the concurrency control program uses the information of the status of data items and transactions, a data item or a transaction was enabled to have variables. These variables are called *control variables*. The operations manipulating control variables are adopted as the functions of program.

A transaction is often related to a data item. The variable binding a transaction to a data item has been introduced. This variable is called the *shared control variable*. All of the operations except for the one setting a data value to a control variable are the terminals of program.

Other terminals include the operations obtaining the identifier of a transaction, and the identifier of a data item, those obtaining a value, e.g., zero, and one, the one of aborting a transaction, and the one making a transaction fall in the waiting state.

The condition evaluation operations were also adopted as functions because flow of program has to be branched according to the values in variables. Functions for concurrency control are also included. Please refer to our previous work [10].

C. Fitness measurement function

Fitness is calculated by using the following six factors: serializability, the degree of concurrency, the response time of a transaction, the abort ratio of transactions, the size of program, and the degree of conflict equivalence.

The re-arranged schedules have to be serializable for ensuring consistency in spite of concurrent execution of transactions. This is very important for concurrency control. When the serializability is not confirmed, the evaluation value is set to zero.

The smaller the program is, the better it is when the program is the same. The size of program is introduced for this measure.

The other four factors are the popular measures of scaling the goodness of the concurrent execution of transactions. Please refer to our previous work [11,10] for further information.

IV. APPROACH

A. Using ADFs

Control variables and shared ones have been introduced in order to attain the generality of programs generated as described before. This means that the programs are generated rather than are combined with some building blocks. This, however, leads the fine grained nodes of program trees. A program must have many nodes in order to have appropriate functionality. For example, the program of 2PL needs 177 nodes. The possibility of the creation of the proper program by combining nodes is very low. The possibility must be increased.

Here, ADFs make the reuse of code possible as described in Section 2. By using ADFs, the same code does not have to be combined many times. The code could be combined as an ADF. This will decrease the number of nodes required for a proper program. ADFs are introduced to program trees.

B. Symbol Sets

1) High-level symbol set: GP is usually used in creating programs from high-level statements [8]. The symbols proposed are not considered to be high-level statements. This makes the program generation difficult. Proper programs are very hard to be generated. The symbols are required to be high-level statements.

To this end, high-level symbols are created by combining conventional symbols, which are functions and terminals. For example, the high-level function *DataValID0isState1* is created by combining conventional functions and terminals. This function invokes the argument if the value of the control variable, whose ID is zero, is equal to one.

a) High-level functions: The functions checking a value of a variable are introduced as the high-level functions. This function invokes the argument specified if the value of the variable whose ID is x is equal to y. Variables are control and shared control ones. The tests include the equality-test and the comparison. In the comparison, a value can be compared with the identifier of a transaction.

We currently limit that the identifiers of control and shared control variables are zero, one, or two, and the values are also zero, one, or two.

The number of the high-level functions is 45.

b) High-level terminals: The terminals setting a value to a variable are also introduced as the high-level terminals. Variables are control and shared control variables. The high-level terminals currently have the same limit as that of the high-level functions.

The number of the high-level terminals is 21.

The total number of symbols becomes 112. The symbol set including the high-level functions and terminals is called the *high-level symbol set*.

2) Reduced symbol set: We have introduced control variables and shared ones. This means that we must prepare two sets of functions for the operations treating two kinds of variables: control ones and shared control ones. This causes the increase of functions. Shared control variables could not be simulated by control variables, while control variables could be simulated by shared ones. We decided not to use control variables. Shared control variables are used instead of control variables. For this purpose, six functions are introduced. For example, one of the functions checks whether the data manipulated by the current operation has a shared control variable whose value is equal to one. Limiting the usage of variables to shared control variables could drastically decrease symbols. The number of symbols becomes 47. This symbol set is called the *reduced symbol set*.

C. Revising the fitness function

The fitness measure function described in Section 3 is very sensitive. The difference between the fitness of the program, which is a little bit effective, and that of the one, which is currently not effective but may become effective, is very large. This prevents the program, which will become effective, from being bred. This kind of program as well as the program, which is not valuable, could not remain. In order to decrease such difference, Equation (7) is adopted.

$$E' = (\log_4 E) + 1.0 \tag{7}$$

The base of a logarithm is determined to be four according to the result of the pre-examination.

V. EXPERIMENTS

A. Procedure

Table I (Table II, respectively) shows the values of the parameters on transactions (genetic operations). The values of the parameters s3 and s12 are varied in the experiment.

Making the number of the values operated (p2) be less than that of transactions (p1) and making the probability of write operations be high result in the situation where conflicts easily occur.

The concurrency control programs tried to be generated are the semantic ones [11]. Semantic concurrency control programs improve concurrency of executions of transactions by considering the semantics of transactions. For example, the operations of the transactions recording logs are the

APPEND operation rather than the UPDATE operation. The APPEND operations do not conflict each other. Whereas this operation is a kind of WRITE operation and usually conflicts other operations, it does not conflict with other operations in this case. By using this characteristic, concurrency could be improved because two APPEND operations could concurrently be executed. The extension for capturing the semantic information on the operations has been introduced in our previous work [11]. The experiment follows this extension.

TABLE I. PARAMETERS FOR TRANSACTIONS

Param.	*Description*	*Value*
p1	Number of transactions	10
p2	Maximum number of data items	3
p3	Ratio of write operations	0.8
p4	Maximum number of operations	10
p5	Probability of abortion of a transaction	0.1

TABLE II. PARAMETERS FOR GP

Param.	*Description*	*Value*
s1	Number of generations	1000
s2	Number of programs a generation	1000
s4	Maximum depth of a program	8
s5	Growing method of a program	random
s6	Probability of crossover of functions	0.1
s7	Probability of crossover of terminals	0.6
s8	Probability of copying a program	0.1
s9	Number of schedules used in the learning	900
s10	Number of schedules used in the evaluation	100
s11	Preference of small programs	0.005
s13	Maximum depth of trees of ADFs	6
s14	Growing method of ADFs	random

The two symbol sets are used in the experiments.

1) Experiments with the high-level symbol set: The following three experiments are conducted with the high-level symbol set.

a) Experiment with initial programs: The programs of the locking protocol, 2PL, and TSO are specified as initial programs. One individual at the initial generation is the initial program specified. As three kinds of programs are specified, three individuals at the initial generation are the programs specified.

b) Experiment without initial program: No initial program is specified.

c) Experiment with initial ADFs: Initial ADFs are set to the ADFs of a half of individuals at the initial generation. Three kinds of ADFs are used as the initial ADFs. These are obtained by decomposing the program of the locking protocols. One is the main process of locking. That is, if the data item is already locked, then the transaction waits for the release of the lock. Otherwise, the transaction gets the lock of the data item. Another is of the termination processing of locking. This releases the variables used. The other is of a kind of control. If a shared control variable is used, the function or terminal specified is executed.

2) Experiment with the reduced symbol set: The experiment is conducted with the reduced symbol set and no initial program.

B. Experimental Result

1) Experiments with the high-level symbol set:

a) Using initial programs: The programs generated are neither degraded from the initial programs, nor greatly evolved. The average of the fitness scores is 5.8. That of 2PL is also 5.8. The highest score is 6.0 for both of the programs.

b) No initial program: The program obtained aborts the transaction if the operation is READ. This program is called the *READ abort program*. The average of the fitness scores is 5.1, while that of 2PL is 5.8.

c) Using initial ADFs: The program generated is almost the same as the READ abort program. This may be caused by that this program is created before the program obtained by combining ADFs is created. As the fitness of this program is high, it is considered that this program could survive. The average of the fitness scores is 4.5. That of 2PL is 5.6. The highest score of the program generated is 5.0, while that of 2PL is 5.9.

2) Experiment with the reduced symbol set: The program could successfully be created. The program created is not a trivial one. The main program only calls an ADF. In the ADF, the value 0 is set to the shared control variable if its value is 2. Next, it is tested whether the data manipulated by the current operation has a shared control variable whose value is equal to two. If so, the transaction is fallen into the WAIT state. If the operation is the last one, all of the shared control variables are deleted. Lastly, the value 2 is set to the shared control variable. The program generated is a kind of the locking protocol. Please note that an ADF is used whereas initial ADFs are not specified. The average value of the fitness scores is 4.3. That of 2PL is 5.9. The highest score of the program obtained is, however, 5.8, while that of 2PL is 6.0. The highest score of the program generated is nearly equal to that of 2PL.

VI. CONSIDERATION

The numbers of symbols and those of the nodes of the program of 2PL are shown in Table III. When the number of symbols is large, the program having a few nodes such as the READ abort program could be generated. It is, however, considered that it is difficult to generate the program large

TABLE III. NUMBERS OF SYMBOLS AND NODES OF PROGRAMS

Symbol set	*Number of symbols*	*Number of nodes for 2PL*
Conventional symbol set	40	177
High-level symbol set	112	72
Reduced symbol set	47	52

enough to have the sufficient functionality of concurrency control. This may be caused by the huge number of combination of symbols due to the large number of symbols. When the number of symbols is less than 50, an appropriate program could be generated as shown in Table III.

The high-level symbols are introduced in order to easily generate appropriate programs. The number of the nodes required for appropriate programs, e.g., 2PL program, could be decreased. On the contrary, the number of the symbols in the symbol set has increased. This results in the explosion of combination of symbols. Introducing high-level symbols is not a good approach.

The number of symbols could drastically be decreased by limiting the usage of variables to the shared control variables. This seems to be a good direction. The program, which may become appropriate, could be generated.

Nodes of program trees that could be randomly selected are used in many applications of GP. The program used in system often proceeds works with managing information. For this end, variables are used. For using a variable in a program, the variable is required to be defined, initialized, manipulated, and often used in the condition evaluation. For example, a counter is used for counting a number of events. When a variable X is used, the variable X must be defined or declared. The variable X must usually be initialized. The value stored in X is incremented when an event occurs. The value stored in X may be used in testing whether the condition is satisfied. In this way, the information is managed with variables. Please note that all of declaration, initialization, manipulation, and utilization are inevitable. Lacking one of them makes the program no meaning. The kind of system program including the concurrency control one requires this kind of combination. It is considered to be very difficult for the combination for the proper program to be generated by randomly combining symbols. Easy acquirement of proper combinations is required.

Although we considered the usage of initial programs effective, it may not be correct. By using initial programs, it may become easy to generate appropriate program. The program generated, however, strongly receives the effect of the initial programs. It seems to be hard for the programs to escape from the initial states.

The fitness measure function is considered to be one of the reasons why appropriate programs could not be generated by using initial ADFs. As READ operations are very few, aborting the transactions including the READ operation may not affect the fitness. In this case, it may be required that aborting transactions leads to low fitness. For this end, weighting terms of the formula of calculating fitness values may be required.

VII. CONCLUDING REMARKS

This paper tried to generate an appropriate concurrency control program by using GP. We used ADFs and introduced high-level symbols in order to make the symbols high-level. Introducing high-level symbols results in increasing the number of the symbols. This makes the program generation difficult. On the other hand, decreasing the number of the symbols caused the generation of appropriate programs. ADFs are also used in the program generated.

By using only shared control variables, every concurrency control program could not be implemented. For example, TSO could not fully be implemented. In TSO, timestamps are added to data items. Adding timestamps to data items is difficult to be implemented with only shared control variables, while it is easy with control variables. Managing information on data items with shared control variables is in future work. The previous section describes that obtaining the proper combination of the operations to variables is required. Easy acquirement of proper combinations is also in future work. Weighting the terms of the formula of calculating fitness values is required as mentioned in the previous section. It is also in future work.

REFERENCES

[1] M. Stonebraker, and U. Cetintemel, "One size fits all : an idea whose time has come and gone," Proc. of the 21st Int'l Conf. on Data Engineering (ICDE2005), 2005, pp. 2-11.

[2] M. Seltzer, "Beyond Relational Databases," Commun. ACM, vol. 51, no. 7, 2008, pp. 52-58.

[3] A. Silberschatz, H. Korth, S. Sudarshan: Database system concepts, Mc-Graw-Hill, 2002.

[4] G. T. Heineman, and G. E. Kaiser, "The CORD approach to Extensible Concurrency Control," Proc. of the 13th Int'l Conf. on Data Engineering (ICDE97), 1997, pp.562-571.

[5] C. Hasse and G. Weikum, "Inter- and Intra- Transaction Parallelism for Combined OLTP/OLAP Workloads," Advanced Transaction Models and Architectures," (S. Jajodia and L. Kerschberg eds.), Kluwer Academic Publishers, 1997.

[6] B. R. Badrinath, and K. Ramamritham, "Semantics-Based Concurrency Control: Beyond Commutativity," ACM Transactions on Database Systems, Vol. 17, No.1, 1992, pp. 163-199.

[7] R. S. Barga, and C. Pu, "A Reflective Framework for Implementing Extended Transactions," Advanced Transaction Models and Architectures," (S. Jajodia and L. Kerschberg eds.), Kluwer Academic Publishers, 1997.

[8] M. Affenzeller, S. Winkler, S. Wagner, and A. Beham, "Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications," CRC Press, 2009.

[9] H. Iba, T. K. Paul, and Y. Hasegawa, "Applied Genetic Programming and Machine Learning," CRC Press, 2010.

[10] S. Tamura, T. Hochin, and H. Nomiya, "Generation Method of Concurrency Control Program by Using genetic Programming," Proc. of 12th ACIS Int'l Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD2011), pp. 175-180, 2011.

[11] S. Tamura, T. Hochin, and H. Nomiya, "Generation of Semantic Concurrency Control Program by Using genetic Programming," to appear in Proc. of 1st IIAI/ACIS Int'l Symposium on Innovative E-Services and Information Systems (IEIS2012), 2012.