# Automatic Generation of Efficient Solver for Query-Answering Problems

Songhao He
Department of Information Science and Technology
Hokkaido University
Sapporo, Japan
E-mail: hesonghao@ist.hokudai.ac.jp

Kiyoshi Akama, Bin Li
Information Initiative Center, Hokkaido University
Sapporo, Japan
E-mail: akama@iic.hokudai.ac.jp
E-mail: zjulb@hotmail.com

***Abstract*—The Query-Answering (QA) problem is a class of the logical problem that is more general than the proof problem and the database searching problem, and can be applied in the semantic web. In this paper, we develop a new technology about how to generate an efficient solver (C program) corresponding to a given QA problem. We expect to generate the specific solver, not the general solver for all kinds of QA problem. The solver is generated based on the bottom-up solution used to update models in the QA problem. We have also developed the technology to suppress the size of the solver to deal with the large-scale QA problem.**

***Keywords-Query-Answering problem; bottom-up solution; specific solver; unfold transformation; support set.***

## I. INTRODUCTION

We know that Description Logic (DL) is a cornerstone of the Semantic Web [1] for its use in the design of ontology, and adding a rule layer on top of the DL-based Web Ontology Language (OWL) is currently the central task in the development of the Semantic Web. In the final analysis, all of these efforts are going to get exactly the right answers of problems, not only judging problems by right or wrong like a proof problem.

The QA problem is a problem whose all answers should be obtained as ground facts. However, adding rules to DL to solve the QA problem in an efficient way is an incomplete research. Moreover, to deal with the logic problem in the real Semantic Web, we should extend the logical expression of the QA problem such as being composed of FOL, DL, Horn clauses, etc. Therefore, the domain of QA problem in our research is larger than the problem defined in the current Semantic Web. An efficient solution corresponding to the QA problem is very important in the development of the Semantic Web. However, a sound solution for QA problems has not yet been satisfactorily established. The process speed of the proposed solution [2] decreases when the size of the problem grows. And it might be impossible to be processed as the size goes beyond a certain size.

In the bottom-up solution proposed by the previous research [2], the data structure of the model is enumeration type, in the pre-model' updating process before generating the representative model, it has to scan the constituted atom from the first atom of the pre-model in the operation like the retrieval, the addition, and the deletion. The calculate speed is not so fast. Moreover, towards a large-scale QA problem the data might explode, and it is possible that the problem cannot be compiled because of the limited memory [3, 4, 5].

In our research, to solve these problems (speed and memory), we generate the specific solver corresponding to the given QA problem by using the specific properties of the problem instead of the general solver for all kinds of QA problem. We think that the entire efficiency improvement could be realized if the processing speed of the specific solver is even fast though it needs to cost the process of generating the solver. Concretely, the specific properties of the problem in this research are clauses of the problem.

Moreover, towards a large-scale QA problem the data size might explode and the computing time takes a lot. Sometimes the problem cannot be solved because of the limited memory. Therefore, we have also developed the technology to suppress the size of the solver to deal with the large-scale QA problem. To generate the efficient solver, simplifying initial clauses obtained from the given QA problem based on the idea of the top down solution named unfold transformation is applied.

The solution of the QA problem proposed in this research is a combination of the top-down solution and the bottom-up solution. And there are four steps in generating the specific solver for the given QA problem.

*A. Clauses of the QA problem are simplified by unfold transformation based on Equivalent Transformation theory*

*B. The set (support set) including all possible ground atoms obtained from clauses are requested in a minimized size.*

*C. A bit array that corresponds to this support set is made.*

*D. Clauses, which are used to update the model, are transferred into "if statement or for loops" in C program as few as possible.*

*E. The final generated C program is the specific solver corresponding to the given QA problem.*

## II. QA PROBLEM AND RESEARCH PURPOSE

### A. QA Problem

In this research, the QA problem is a more general class of the logical problem than the proof problem and the database searching problem, and can be applied in the semantic web.

 ISBN: 978-1-61208-148-9 

The QA problem contains knowledge ($\Delta$) and the query atom ($q$), in which $\Delta$ not only includes the definite clause (atom0 ← atom1, atom2…), which means the clause has only one atom in the left side of the arrow, but also the negative clause (← atom1…), which means there is no atom in the left side, and non-definite clause (atom0, atom1, … ← atom2, atom3…), which means there are more than one atom in the left side. In this research, the QA problem is described by the logical expression and we have to obtain all answers to it. It is possible to describe such problem's answer generally as follows.

$$A = \{g|\Delta|=g \in G, g = q\theta, \theta \in S\}$$

Here, we take the *Oedipus* [6] problem as an example.

> *"OE is the child of IO. PO is the child of IO. PO is the child of OE. TH is the child of PO. OE is a patricide. TH is not a patricide. A person's child is a patricide, but his/her grandchild is not a patricide. Who is the person?"*

This problem is composed of knowledge ($\Delta$, "OE is the child of IO. …, but his/her grandchild is not a patricide.") and the question ($q$, "Who is the person?"). The result ($A$) is "IO". This QA Problem can be rewritten as the following clauses (There are two atoms existing in the left side of the arrow including the question atom, which indicates to wider meaning QA problem).

***Knowledge (Δ):***
isChild(oe, io) ←. isChild(po, io) ←. pat(oe) ←.
isChild(th, po) ←. isChild(po, oe) ←. ← pat(th).
prob(*x),pat(*b)←isChild(*a, *x),pat(*a),isChild(*b, *a).
***Query Atom (q):*** prob(*x)
***Answer (A):*** {($\Delta$, $q$)} ==> {io}.

### B. Research Purpose

In this research, we want to develop a new technology about how to generate an efficient solver (C program) corresponding to a given QA problem. Before generating the specific solver (C program) for a given QA problem, not the general solver for all kinds of QA problems, we need to do the memory saving work. Therefore, the present research purpose is shown as follows.

1) *As the size of the QA problem grows, suppressing the memory consumption is important. To generate the efficient solver, the reduction of the size of the QA problem by simplifying initial clauses based on the idea of the top down solution named unfold transformation is applied.*

2) *We propose how to generate the solver corresponding to the given problem by using the unfolded clauses based on the bottom-up solution. Because the final solver is generated by C program, the data structure of the model is the bit-array type, not the usual enumeration type.*

## III. APPROACH OF THE RESEARCH

To achieve the research purpose, we do the following four steps (Figure 1) [7, 8].

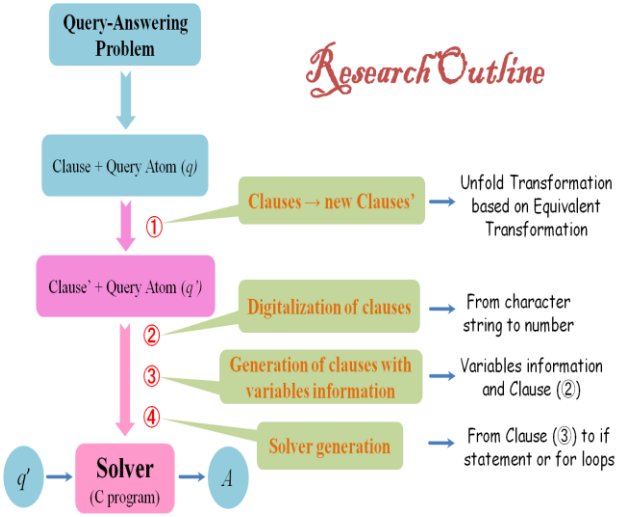


Figure 1. Approach of the research

### A. Simplification Processing of Clauses based on the Unfold Transformation

In this research, as the size of the QA problem grows, the memory consumption for updating the model will become very big. Before applying the clauses obtained from the QA problem, we firstly do the unfold transformation based on the equivalent transformation theory, the simplification of clauses is pursued by substituting the definite clause, which will be introduced in Section 4.

### B. Digitalization of Clauses

As introduced in the research purpose, we want to generate the specific solver (C program) for each QA problem, it is necessary to make the mechanism about how to convert clauses, which are the result of the unfold transformation in step A, into the corresponding C program. In order to transfer clauses to C program, the algorithm about how to convert the atom, the basic element consisting of the clause, into the index number of the bit array used in C program is very important.

### C. Generation of Clauses with Variables Information

When we convert each clause into C program (if statement or for loops) by using the result of process B, in order to consolidate the size of the generated C program, we want to generate for loops as far as we can. For this reason, we need to firstly obtain the variables information of each clause (the information of clauses without variables is empty, and this kind of clause will be transferred into if statement), then based on this variables information, the clause with variables will be converted to for loops.

 ISBN: 978-1-61208-148-9 

### D. Solver Generation

Based on atoms-index's corresponding algorithm obtained in step B and clauses with variables information generated by step C, all clauses will be transferred into the corresponding if statement or for loops in C program. As a result, the solver corresponding to the given QA problem is generated. Finally, the answer of the given QA problem will be got by executing the generated solver.

In Section 4, we will introduce the simplification processing of clauses based on the unfold transformation. In Section 5, we will explain the processing of digitalization of clauses. In Section 6, generation of clauses with condition and automatic generation of solver will be introduced.

## IV. SIMPLIFICATION PROCESSING OF CLAUSES BASED ON THE UNFOLD TRANSFORMATION

To apply a large-scale QA problem, the size and the complexity of clauses requested from the QA problem can be reduced by the unfold transformation based on the Equivalent Transformation theory [9, 10, 11].

In this research, the unfold transformation is started by deciding the target predicate of the atom, and other atoms (with different predicates) exist in the same clause would be substituted by definite clauses. The definite clause will be finally removed after being applied. In this way, the simplification of clauses obtained from the given QA problem could be accomplished.

Here, there is an example showing the unfold transformation for a non-definite clause (e.g.:← (Wolf *A) (Fox *B) (eat *A *B).), which is done by using two ground clauses (e.g.: (Wolf wolf) ←. (Fox fox) ←.) (Figure 2).

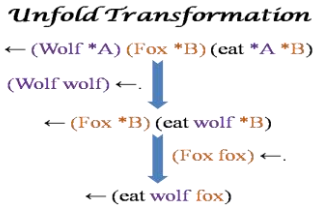


Figure 2. Process of the unfold transformation

## V. DIGITALIZATION AND LIMITATION OF CLAUSES

### A. Generation of the Support Set

In the process of generating the solver from clauses, we need to first decide the set of all possible atoms which constitute clauses. In this research, the set of all atoms are called the support set. The digitalization of clauses can be made by deciding the support set. The support set requesting process has been divided into three steps (A, B, C) shown in Figure 3. Clauses requested by the unfold transformation is input, and the support set corresponding to the problem is output. In the approximate processing of A, the atom corresponding to the problem is roughly requested. The smaller and more accurate the support set requested, the higher calculation cost for generating the support set is. Based on the approximate idea, we do not look for the most accurate support set, but within an approximate range, search a little wide-ranging support set efficiently, and finally generate the program used to request the support set. The support set is requested by executing the program.

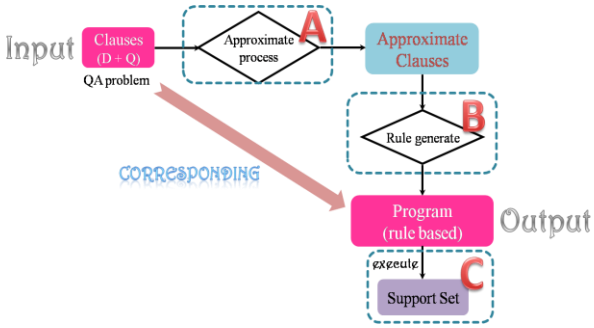


Figure 3. Process of generating the support set

*1) Approximate Process (A)*

Clauses in the Oedipus problem:

| | |
|---|---|
| (isChild oe io) ←. (isChild po io) ←.<br>(isChild po oe) ←. (isChild th po) ←.<br>(pat oe) ←. ← (pat th).<br>(prob *x), (pat *b)<br>← (isChild *a *x), (pat *a), (isChild *b *a). | ***Input*** |

New generated clauses:

| | |
|---|---|
| (isChild1 oe) ←. (isChild2 io) ←.<br>(isChild1 po) ←. (isChild2 oe) ←.<br>(isChild1 th) ←. (isChild2 po) ←.<br>(pat oe) ←. ← (pat th).<br>(prob *x) ← (isChild1 *a), (isChild2 *x), (pat *a),<br>(isChild1 *b), (isChild2 *a).<br>(pat *b) ← (isChild1 *a), (isChild2 *x), (pat *a),<br>(isChild1 *b), (isChild2 *a). | ***Output*** |

*2) Generation of the Approximate Clauses (B)*

| | |
|---|---|
| 1: (isChild1 oe) ←. 2: (isChild2 io) ←.<br>3: (isChild1 po) ←. 4: (isChild2 oe) ←.<br>5: (isChild1 th) ←. 6: (isChild2 po) ←.<br>7: (pat oe) ←. 8: ← (pat th).<br>9: (prob *x) ← (isChild1 *a), (isChild2 *x),<br>(pat *a), (isChild1 *b), (isChild2 *a).<br>10: (pat *b) ← (isChild1 *a), (isChild2 *x),<br>(pat *a), (isChild1 *b), (isChild2 *a). | ***Input*** |

***Output***:

(whole *isChild1 *isChild2 *pat *prob *x1 *x2 *x3 *x4), {(addelem *isChild1 (oe) on *isChild1new)}→(whole *isChild1new *isChild2 *pat *prob *x1 *x2 *x3 *x4). ➔1

…

(whole *isChild1 *isChild2 *pat *prob *x1 *x2 *x3 *x4), {(inter (*pat *isChild2 *isChild1) *mid1), (inter (*isChild2) *mid2), (inter (*isChild1) *mid3), (cons *mid1), (cons *mid2),

(cons *mid3), (addelem *pat *mid3 on *patnew)}→(whole *isChild1 *isChild2 *patnew *prob *x1 *x2 *x3 *x4). ➔10

*3) Generation of the Support Set by Executing the Rule Based Program (C): By applying rules, each predicate set (isChild1, isChild2, pat, prob) that constituted the support set has been expanded (Figure 4).*

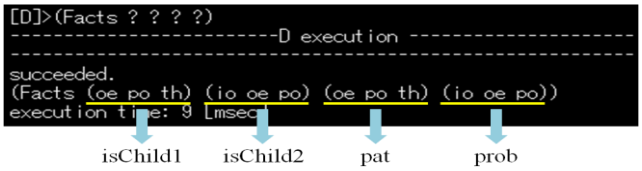


Figure 4. Generating of each predicate set

The support set is composed by combining these predicate sets, and the support set of the Oedipus problem is shown as follows.

{(isChild oe io), (isChild oe oe), (isChild oe po), (isChild po io), (isChild po oe), (isChild po po), (isChild th io), (isChild th oe), (isChild th po), (pat oe), (pat po), (pat th), (prob io), (prob oe), (prob po)}

Expression in clauses (Figure 5) :

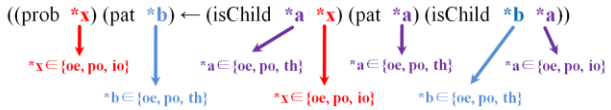


Figure 5. Support set expressed in the clause

## B. *Digitalization of the Support Set*

First of all, for all atoms in the support set, the symbol set that includes all the symbol values which can substitute the argument are requested. The order of symbols in the symbol set is decided, and each symbol is converted into the natural number. Second, all atoms in the support set are sorted in alphabetical order of the argument by the order of this symbol set (Figure 6).

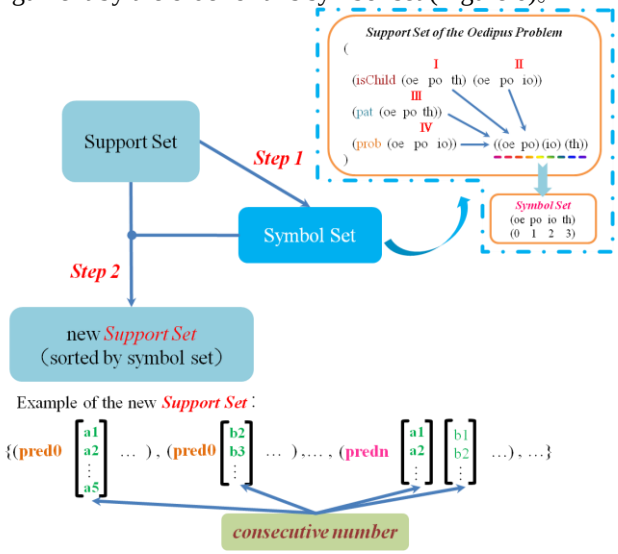


Figure 6. Digitalization of the support set

Symbol Set:

(oe po io th) → (0 1 2 3)

***Input:***

{(isChild oe io), (isChild oe oe), (isChild oe po), (isChild po io), (isChild po oe), (isChild po po), (isChild th io), (isChild th oe), (isChild th po), (pat oe), (pat po), (pat th), (prob io), (prob oe), (prob po)}

***Output:***

{(isChild 0 0), (isChild 0 1), (isChild 0 2), (isChild 1 0), (isChild 1 1), (isChild 1 2), (isChild 3 0), (isChild 3 1), (isChild 3 2), (pat 0), (pat 1), (pat 3), (prob 0), (prob 1), (prob 2)}

Expression in clauses (Figure 7) :

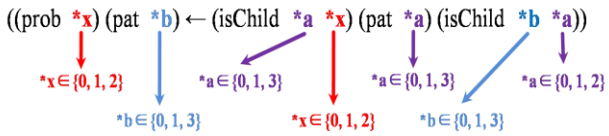


Figure 7. Digitalization of support set expressed in the clause

## C. *Digitalization of Clauses*

In this research, we will finally convert each clause into the corresponding the C program (if/for loops), it is necessary to make the mechanism about how to convert the atom into the address of the bit array. Here, the atom-address calculating function, used to make all basic atoms in the support set correspond to the address of the bit array, is made. Based on the function, it is possible to access the address which corresponds to the atom quickly in the updating process.

In the sorted support set (from the result of B), the argument of atoms with a consecutive value is brought together. Then, the address function corresponding to this kind of atom is generated. It generates completely different address function for discontinuous argument value in spite of having the same predicate (Figure 8).

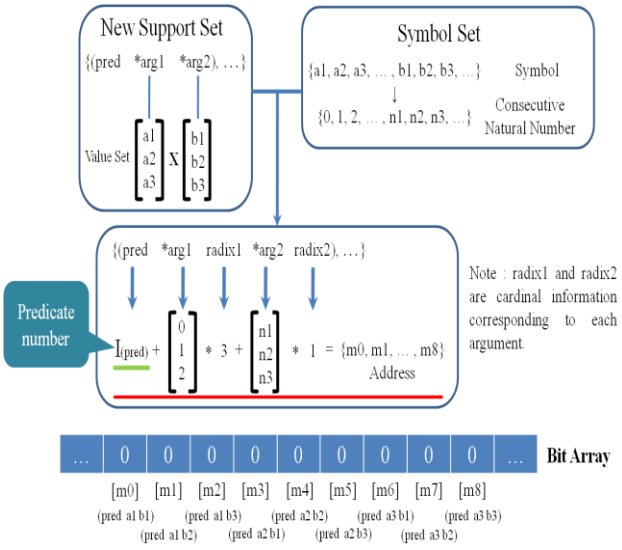


Figure 8. Digitalization of clauses

The address "PAdr(s1,...,sn)" of all basic atoms "Pred(s1,...,sn)" (There are n arguments) can be decided by the introduced algorithm. It is requested from the predicate number "I(pred)", and the relative address "Rel(s1,...,sn)" of the predicate "Pred", based on the following formula.

$$PAdr(s1,...,sn) = I(pred) + Rel(s1,...,sn) \quad (1)$$

Relative address "Rel(s1,...,sn)" is requested from the 1st argument value "Sym(s)" and its cardinal "R(pred,s)" of the last argument by the following formula.

$$Rel(s1,\ldots,sn)=Sym(s1)*R(pred,s1)+\ldots+Sym(sn)*R(pred,sn) \quad (2)$$

Cardinal "R(p,i)" is requested by using symbol number "Ssym(p, k)". For instance, based on the support set how many symbols can substitute the back arguments.

$$R(p,i) = \prod_{k=i+1}^{n} S_{sym}(p,k)$$

By applying the above-mentioned function, atoms in the clause can be conver into address of the bit-array.

Expression in clauses (Figure 9) :

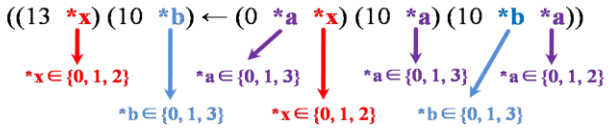


Figure 9. Example of digitalization of clauses

## D. Limitation of Clauses

Based on the result of clauses requested in process C, the atom with the same variable is extracted, and the intersection calculation of the value set of the argument is done. The result is shown in Figure 10.

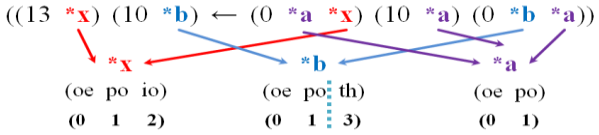


Figure 10. Example of Limitation of clauses

As shown in Figure 10, because the value set corresponding to the variable (*b) is not a consecutive value, the clause with condition is generated based on the value set corresponding to the argument (limitation of clauses). The condition part of ground clauses is empty.

eg.: {(*x 0 2) (*b **0 1**) (*a 0 1)}

((13 *x) (10 *b) ← (0 *a *x) (10 *a) (0 *b *a))

{(*x 0 2) (*b **3**) (*a 0 1)}

((13 *x) (10 *b) ← (0 *a *x) (10 *a) (0 *b *a))

# VI. SOLVER GENERATION

## A. Solver Generation

In this research, the solver is composed by three parts, which are main function definition, bit-array declaration and if/for loops. We input the query (*q*), the solver will output the corresponding answer (*A*). The generation of if/for loops is requested by using the conditional clause generated in the process D of Section 5. Here, we will introduce the method about how to convert various conditional clauses into if/for loops of C program.

### *1) From Ground Clauses to if loops*

The transmission from a ground clause to an if loops in C program is shown as follows (Figure 11).

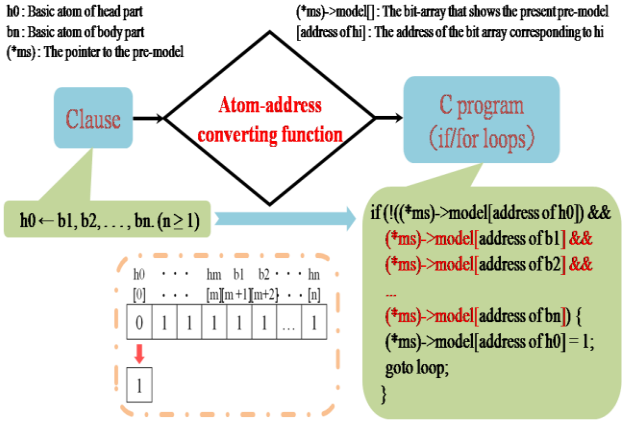


Figure 11. From ground clauses to if loops

### *2) From Not Ground Clause to for loops*

A clause that contains variables will first substitute all variables for possible symbols, and then generate new ground clauses. The patterns of the symbol those can substitute the variable increase while the size of the QA problem grows. Therefore, the size of the generated C program will grow, sometimes it will become impossible to compile.

In this research, because the value set corresponding to the variable of each atom in the clause is obtained based on the support set, and been changed into natural numbers (starting from 0) based on the symbol set, the consecutive value will be expressed by one "for loops". The solver result of this "for loops" is already shown in Figure 12.

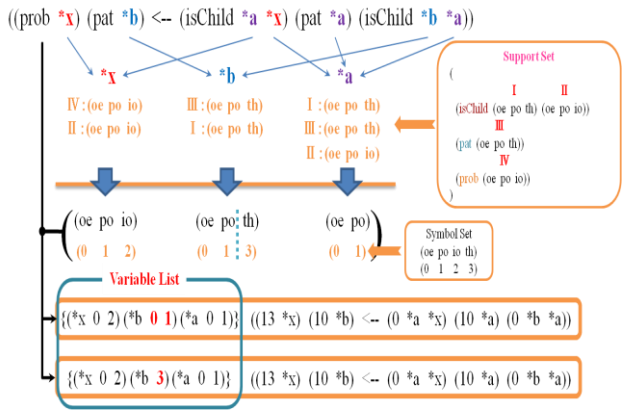


Figure 12. Consolidate processing by "for loops"

*3) Example of Solver Generation*

Here is an example of generating the solver corresponding to the Oedipus problem. Chiefly three parts is shown (Figure 13)。

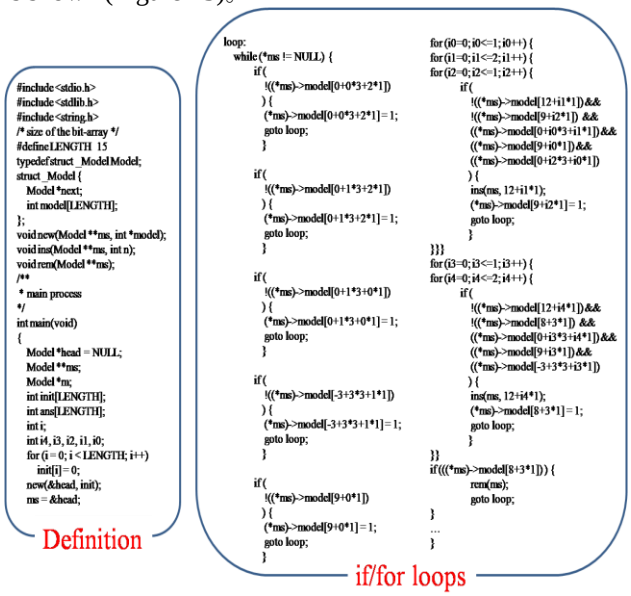


Figure 13. Example of solver generation

## VII. SUMMARY

In this paper, we talk about generating a special solver (C program) for a given QA problem. The solver is composed by main function definition, bit-array declaration, and if/for loops. We input the query (*q*), the solver will output the corresponding answer. Because models generated by the solver are based on the bit calculation, the speed is absolutely fast. We also develop the technology for suppressing initial clauses by the equivalent transformation process.

As future work, we think the answer of the QA problem can be more quickly obtained in a smaller search space by simplifying process of clauses obtained from the QA problem before clauses being used to update the pre-model.

## REFERENCES

[1] G. Brewka: Well-founded semantics for extended logic programs with dynamic preference, Journal of Artificial Intelligence Research, 4, pp. 19-36 (1996).

[2] C. Zheng, K. Akama, and T. Tsuchida: SOLVING "ALL-SOLUTION" PROBLEMS BY ET-BASED GENERATION OF PROGRAMS, International Journal of Innovative Computing, Information and Control, Volume 5, Number 12(A), pp. 4583-4595, 2009.

[3] R. Manthey and F. Bry: SATCHMO: a theorem prover implemented in Prolog. Proc. of the 9th Int. Conf. on Automated Deduction, LNCS 310, pp. 415-434 (1988).

[4] M. Koshimura, H. Fujita, and R. Hasegawa, MGTP: A Model Generation Theorem Prover -Its Implementation and Application-, Kyoto University Research Information Repository, 1125, PP. 65-80 (2000).

[5] F. Bry and A. Yahya, Minimal model generation with positive unit hyper-resolution tableaux, Lecture Notes in Computer Science, 1996, Volume 1071/1996, pp. 143-159.

[6] F. Baader and D. Calvanese et al. The Description Logic Handbook. Cambridge Univ Press, 2001.

[7] S. H. He and K. Akama, Generation of Smaller Programs For Efficient Solution of Query-Answering Problems, The 5th International Conference on Computer Sciences and Convergence Information Technology (ICCIT2010), Proceeding: pp. 665-670, Seoul, Korea, Nov 30-Dec 2, 2010.

[8] S. H. He and K. Akama, Speed-up of the Solution of Query-Answering Problem, The 2010 International Congress on Computer Applications and Computational Science (CACS 2010), Proceeding: pp. 852-855, Singapore, Dec 4-6, 2010.

[9] T. Kawamura and T. Kanamori, Preservation of stronger equivalence in unfold/fold logic program transformation Theoretical Computer Science, Volume 75, Issues 1-2, 1990, pp. 139-156.

[10] K. Akama and E. Nantajeewarawat. Meaning-preserving skolemization on logical structure. Proceedings of the 9th International Conference on Intelligent Technologies (InTech'08), pp. 123-132, 2008.

[11] K. Akama, E. Nantajeewarawat, and H. Koike. Program generation in the equivalent transformation computation model using the squeeze method. In Proc. Of PSI2006, LNCS4378, Springer-Verlag Berlin Heidelberg, pp. 41-54, 2007.

 ISBN: 978-1-61208-148-9 