An Ameliorated Methodology for the Abstraction and Minimization of Functional Dependencies of legacy 'C' Program Elements

Dr. Shivanand M. Handigund
Dept. of Computer Science & Engineering.
Bangalore Institute of Technology
Bangalore –560 004

Rajkumar N. Kulkarni
Dept. of Information Science & Engineering
Ballari Institute of Technology & Management
Bellary – 583 104

ABSTRACT
Information systems of many organizations are processed through system of interrelated 'C' programs. In a 'C' program each statement contains functional dependencies amongst the attributes. Sometimes these functional dependencies may redundant in different statements. The existing application programs used in the maintenance of the information system are lengthy, and because of the perennial maintenance of the program, these functional dependencies are unevenly scattered. Thus, some functional dependencies may be implicitly present in other statements or sometimes they are unevenly scattered across the entire program. This situation creates the complication in the reengineering process which creates scuffle in selecting the attributes for a class on the basis of the cohesive property.

While abstracting the object structures, and making the cohesive groups of attributes, the recursive implicit containment of one functional dependency within another creates complication in the granularity of design elements as the implicit dependencies have ripple effect on the dependencies of attribute. This paper attempts to propose the identification of functional dependencies from the realization of program code, and their minimization through the minimal cover process. The correctness and completeness of the abstraction is a straight forward process.

Keywords
Functional dependencies, minimization, abstraction, reengineering, business rules, legacy systems, reverse engineering.

1. INTRODUCTION
Information systems of many organizations are processed through system of interrelated 'C' programs. Since, the 'C' programming language was developed in the early second half of the last century; it couldn't incorporate to facilitate the current day's state of the art technology. Therefore, the programs developed based on legacy 'C' systems are not coping with the advancement of technologies in the areas of Storage, Processing, Graphical user Interfaces. Moreover since these programs have undergone perennial updations; and as a result these may be unstructured or containing with irrelevant documentation. However the legacy 'C' system contains accumulated business information of the entire organization. There is a need to harness the state of the art technology with the useful business information buried across these legacy 'C' systems. On one side it contains business information accumulated over the years, and on the other side, it is not coping up with the advancement of the technologies. As a result, harnessing the state of the art technology with the existing system is a labyrinth. The following are the different ways to resolve labyrinth [19].

- The first approach is to discard completely the old system and develop a totally new system. This discarding approach suffers from a pitfall of loosing useful business rules accumulated throughout the development and maintenance process. Thus, it will not serve the purpose of developing the new system by incorporating the buried business rules.
- The second approach is to translate the existing 'C' code into the target language code. This translation process can be carried out in two different ways viz. manual and automatic approach. The manual approach is time consuming and error prone approach. Moreover, the enormity of the stored legacy system compels the translation process next to impossible. The automatic translation can be performed by developing a translator tool, because of the flexibility involved in the program coding; the correct and complete translator can't be developed. Thus, the developed translator may not translate the entire code; it may translate the simple code leaving human to translate the complex code. Thus, the translation of the legacy system directly to the target language may not be a good solution for the reuse of legacy 'C' systems as it compels the human intervention. Moreover the translation always takes the taste of the original language.
- The third approach is the wrapping. In this approach, the old system is wrapped through the use of emulators so that at the front end the new system is running but at the back end the same old system is running with the slow phase. Thus, the wrapping approach is also not so useful for the reutilization of the existing code.

- The best alternate approach is the reverse engineering of the old system to the text document and then the text document is reviewed with the state of the art technology requirements. Then through forward engineering it can be transformed to the required target language.

We adopt the last option with some minor modification to propose a new methodology in which we abstract the view elements of the new system from legacy 'C' system by blending the reverse engineering of design process with the redesign of the forward engineering.

2. TAXONOMY

Referenced Attribute: A variable is said to be referenced in a statement if the value of that variable is used during the execution of the statement without getting itself modified. For ex., A = B + C. The values of B & C are used or referenced in the statement [1, 16].

Defined Attribute: A variable is said to be defined, in a statement if the execution of that statement can alter the value of the variable. For ex., A = B + C. The values of B & C are referenced or used in the statement, and A is said to be defined [1, 16].

Preserved Attribute- A variable whose value is unaltered with the execution of statement, then the attribute value is preserved in that statement [1]. A variable may be both referenced and defined in a statement, or may be both preserved and referenced, but cannot be preserved and defined in a statement.

Functional Dependency: If R is an entry in the data flow graph and A and B are non-empty sets of attributes in R, then B is functionally dependent on A, iff A is referenced and B is defined in that entry, and the formal notation would be A → B.

Minimal Cover: The set of dependencies contain or may contain some implicit dependencies embedded in other dependencies. If these implicit dependencies are eliminated the remaining functional dependencies set contain the bare minimum dependencies required for the information system. Minimal cover is the process of eliminating these embedded implicit dependencies.

Legacy Software: The software which is 10 to 15 year old and is the result of perennial need based updations which serves the purpose but resists for modification.

Mission Critical: The Information System cannot run without the presence of software even for a short period.

Abstraction Level: The abstraction level is defined with respect to the proximity to the machine understanding. The abstraction levels (Requirement, Design, Implementation) corresponds to a phase in the software development life cycle and defines the software system at a particular level of detail [1].

Software maintenance: The modification of a software product after delivery to correct faults to improve performance or to satisfy changing business needs [1].

Restructuring: Restructuring is the transformation from one representation form to another at the same abstraction level. The transformation preserves the external behavior of the system. Restructuring here is used in implementation stage to transform code from an unstructured form to a structured form [1].

Reverse Engineering: The reverse engineering is the process of analyzing the subject system with two goals [1]:
- To identify the system's components and their interrelationships
- To create representations of the system in another form at a higher abstraction level.

Forward engineering: Forwarding engineering is the traditional process of moving from the requirements of the system to its design stage, and from design stage to the concrete implementation of the system. It should be preceded by the reverse engineering, in the absence of which it is called software development [1].

Reengineering: The process of re-engineering can be defined by the simple formula [1]:
Re-engineering = Reverse engineering + Change in techniques to suit the new target environment + Forward engineering

3. PROPOSED METHODOLOGY

'C' programs do not have strict indentation rules. Because of the flexibility of indentation in the 'C' programming language multiple statements can be placed on a single line or a single statement can span several lines [18]. All 'C' statements must be terminated with a semicolon and the last statement in the body of the loop may terminate with right brace. Similarly the beginning of the statement follows either semicolon or a blank or left brace. So, in the beginning of the process, we are placing strict indentation rule such as one line one statement, then assigning the consecutive line numbers to logically related statements of the 'C' program except for the blank lines and the comment lines [12, 13, 14, 15]. This serves as a moulded input for the next steps in the abstraction of functional dependencies.

The methodology for the abstraction of functional dependencies and the design of object structures from the 'C' program is explained in the following steps:

3.1 Abstraction Of Control Flow Graph From 'C' Program

The Control Flow Graph (CFG) indicates the execution control flow of the entire program or system of programs. In 'C' program, each line consists of one statement or control predicate. To represent it as a graph we have to assign one vertex (node) for each statement. The control flow between the two logically consecutive statements (but not physically consecutive) is indicated by an edge between those vertices. Since, the Legacy 'C' system contains thousands of lines of code; the CFG may overflow the available memory. To reduce the utilization of memory space, the groups of statements which are both logically and physically in the control flow order are collapsed to a single node. This CFG is stored in buffer in the form of Control Flow table, in which the first and second columns contain statement numbers of start and end statements of the node. The third and fourth columns contain statement numbers of alternate control transits. Here, IF, IF-ELSE, FOR, SWITCH, WHILE, DO-WHILE, EXIT, RETURN, CONTINUE, and FUNCTION CALLS are treated as verbs to enable the parser to identify the beginning of each statement separately. The end of the program is represented with the statement number followed by the character **E**.

We have already developed the automated methodology in the form of a software tool [14, 15]. The CFG of the sample C program depicted in **Figure 1** is shown in **Table 1** as follows:

Table 1.The control flow table for the program in figure 1

START	END	Transition 1 / Next Jump	Transition 2 / Alternate Jump
1	14	15	19
15	18	14	
14	14	15	19
19	23	24	33
24	32	23	
23	23	24	33
33	34**E**		

3.2 Abstraction of Data Flow Graph from 'C' Program

The above designed CFG is used to design Data Flow Graph (DFG) in the form of Data Flow Table (DFT). If the control flow entry doesn't contain the same numbers in the first two columns, then for each consecutive number between the two numbers there is a consecutive entry in the data flow table. The sample entries are shown in the **Table 2**. The data items in a 'C' program are defined by scanf, fscanf, gets, fread statements, the left side attribute of the arithmetic expressions except stdin, and stdout. The data items in a 'C' program are referenced by printf, fprintf, puts, fwrite, statements, the right side attributes of the arithmetic expressions and the attributes of control predicates. The DFG for our sample 'C' program depicted in figure 1 is represented in **Table 2**.

Table 2. The data flow table for the program in figure 1.

Statement Number	Referenced variable	Defined variable
1	---	---
2	---	---
3	---	---
4	---	---
5	---	---
6	---	---
7	---	---
8	---	---
9	---	---
10	---	---
11		Filename
12	Filename	Fp
13	---	---
14	I	i
15	---	---
16		ENAME,BS,LIC, PT,IT,OA
17	ENAME, BS, LIC, PT, IT, OA	
18	---	---
19	Fp	
20	---	---
21	Filename	Fp
22	--	---
23	I	i
24	---	---
25		ENAME, BS,LIC,PT,IT,OA
26	BS	HRA
27	BS	DA
28	BS,DA,HRA,OA	GS
29	LIC,PT,IT	DED
30	GS,DED	NS
31	BS, DA, HRA, LIC, PT, IT, OA, GS, DED, NS	
32	---	---
33	Fp	
34	---	---

3.3 Abstraction Of Functional Dependencies From 'C' Program

If the statement in the Table-2 contains both referenced and defined items, then the referenced items determine the defined items. Thus there is a functional dependency between the referenced and defined items. The proposed tool developed here abstracts the functional dependencies from the input 'C' program shown in **Figure-1.** The functional dependencies abstracted from the program are:

BS → HRA
BS → DA
BS, DA, HRA, OA → GS
LIC, PT, IT → DED
GS, DED → NS
Filename →fp

4. ALGORITHM FOR MINIMAL COVER

Input: Functional Dependencies abstracted from the program
Output: Minimized or minimal cover of Functional dependencies

1. Sort all the Functional Dependencies in the Descending Order of number of attributes of LHS
2. Transform the Functional Dependencies in the canonical form
3. Form a two dimensional table $\mathbf{T_{ij}}$ where **i** represents functional dependency and **j** represents the position of the participating attribute.

4. Fill up all the entries in the table
   If $t_{ij} \in$ LHS of Functional Dependency
   Then $\mathbf{a_{ij} \leftarrow t_{ij}}$ **and** $\mathbf{d_i \leftarrow RHS_i}$

   $\forall$ i ∈ Functional Dependencies

   $$t_{ij} = \begin{cases} a_{ij} & \text{if j is present in LHS of } i^{th} \text{ FD} \\ 0 & \text{Otherwise} \end{cases}$$

   Repeat step 5
   for **k = 1 to n** where k is $\mathbf{k^{th}}$ row that represents $\mathbf{k^{th}}$ functional dependency

5. For i = k + 1 to n
   Consider t_{ij}
   do
   If $\forall$ $\mathbf{a_{ij}}$ ∈ LHS of $\mathbf{i^{th}}$ Functional Dependency
   $\mathbf{a_{ij} = a_{kj}}$

Delete $\mathbf{i^{th}}$ dependency from the further test for retainment
Exclude $\mathbf{k^{th}}$ dependency from further retainment test

The set of retained functional dependencies forms the Minimal Cover.

5. CASE STUDY

5.1 The proposed procedure is implemented for number of 'C' programs and the results we got are correct and complete. The sample 'C' program depicted in figure 1 is the output of moulding process.

```
 1  #include <stdio.h>
 2  #include <conio.h>
 3  main()
4       {
 5  FILE *fp;
 6  int number, quantity,BS,DA,HRA,OA,LIC,PT,IT,GS,
DED,NS,i;
 7  float price,value;
 8  char ENAME[10], filename[10];
 9  clrscr();
 10 printf("Input file name\n");
 11 scanf("%s", filename);
 12 fp = fopen(filename, "w");
 13 printf(" ENAME   BS   LIC   PT   IT   OA\n");
 14 for(i=1; i<=3; i++)
15 {
16 fscanf(stdin, "%s %d %d %d %d", ENAME, &BS,
&LIC, &PT, &IT, &OA);
17 fprintf(fp, "%s %d %d %d %d %d", ENAME, BS, LIC,
PT, IT, OA);
18          }
19  fclose(fp);
20  fprintf(stdout, "\n\n");
21  fp = fopen(filename, "r");
22  printf("ENAME    BS  DA  HRA  LIC  PT  IT  OA
GROSS  DED  NET_SAL \n");
23  for(i=1; i<=3; i++)
24 {
25 fscanf(fp, "%s %d %d %d %d %d", ENAME, &BS, &LIC,
&PT, &IT, &OA);
26       HRA = BS * 0.09;
27       DA = BS * 1.14;
28       GS = BS + DA + HRA + OA;
29       DED = LIC + PT + IT;
30       NS = GS - DED;
31  fprintf(stdout, "%s %d %d %d %d %d %d %d %d
%d%d", ENAME,  BS, DA, HRA, LIC, PT, IT, OA, GS,
DED, NS);
32  }
33   fclose(fp);
34  }
```

Figure 1. A sample 'C' Program

The output of the steps 1- 4 of the algorithm mentioned in section 4 is shown in Table-3. The step 5 is applied to the table, but the table entries are in the normalized form only. The dependencies cannot be further minimized. So the minimal cover or minimized functional dependencies in the abstracted functional dependencies are:

BS, DA, HRA, OA → GS
LIC, PT, IT → DED
GS, DED → NS
BS → HRA
BS → DA
Filename →fp

5.2 Consider another set of functional dependencies given below:

1. A B C D → E F
2. A → F
3. B → G
4. A B D → E F
5. A C D → F G

Apply all the steps of the Algorithm mentioned in Section – 4 of the paper. The result is shown in **Table – 4**.
The output generated by the automated tool is shown below:

1. A → F
2. A B D → E
3. A C D B → G

The attributes of RHS grouped on the same set of LHS **attribute form group of attributes for a method.**

Table -3: minimization of functional dependencies for the case study 5.1

FD No.	***BS***	***DA***	***HRA***	***OA***	***LIC***	***PT***	***IT***	***GS***	***DED***	***NS***	***Filename***	***Dependency***
1	a_{11}	a_{12}	a_{13}	a_{14}	0	0	0	0	0	0	0	GS
2	0	0	0	0	a_{25}	a_{26}	a_{27}	0	0	0	0	DED
3	0	0	0	0	0	0	0	a_{38}	a_{39}	0	0	NS
4	a_{41}	0	0	0	0	0	0	0	0	0	0	HRA
5	a_{51}	0	0	0	0	0	0	0	0	0	0	DA
6	0	0	0	0	0	0	0	0	0	0	a_{61}	Fp

Table-4.Minimization of functional dependencies for the case study 5.2

FD No	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**Dependency**
1	a_{11}	a_{12}	a_{13}	a_{14}	0	0	0	E
2	a_{21}	a_{22}	a_{23}	a_{24}	0	0	0	F
3	a_{31}	a_{32}	0	a_{34}	0	0	0	E
4	a_{41}	a_{42}	0	a_{44}	0	0	0	F
5	a_{51}	0	a_{53}	a_{54}	0	0	0	F
6	a_{61}	0	a_{63}	a_{64}	0	0	0	G
7	a_{71}	0	0	0	0	0	0	E
8	a_{81}	0	0	0	0	0	0	G

6. CONCLUSION

This paper presents an automatic tool that abstracts attributes and forms the first cut object classes through the abstraction of functional dependencies. This procedure is implemented by a scenario of methods to abstract control flow graph and then to represent the data flow graph in the form of data flow table. Using the data flow table of the program and data & control dependencies concept, the functional dependencies are linked to form the attributes closure. Repeating this procedure, different functional dependency sets are obtained, and then the functional dependencies are minimized using the ameliorated algorithm. The attributes within each functional dependencies group forms the first cut object structures. The inheritance property can be well represented if the implicit dependencies are eliminated. This structure is based on the formation of cohesive attributes using good database and software engineering principles.

7. REFERENCES

[01] Shivanand M. Handigund, "Reverse Engineering of Legacy COBOL systems", Ph.D. Thesis, 2001, IIT Bombay, Mumbai

[02] Ronald S. King, James J. Legendre, "Discovery of Functional and Approximate Functional Dependencies in Relational Databases", Journal of Applied Mathematics And Decision Sciences, 7(1), 49-59, 2003.

[03] Wie Ming LIM, John Harrison, "Discovery of constraints from data for Information system Reverse Engineering", IEEE 1997, 39-48.

[04] Wie Ming LIM, John Harrison, "Parallel approaches for Discovering Functional Dependencies from Data for Information System Design Recovery", IEEE 1997, 254-260.

[05] Victor Matos, Becky Grasser, "SQL-based Discovery of

Exact and Approximate Functional Dependencies", SIGCSE Bulletin, Volume 36, Number 4, Dec-2004, 58-63.

[06] Hong Yao, Howard J. Hamilton, and Cory J. Butz, "FD_Mine: Discovering Functional Dependencies in a Database Using Equivalences".

[07] Jalal Atoum, Dojanah Bader, and Arafat Awajan, "Mining Functional Dependency from Relational Databases Using Equivalent Classes and Minimal cover", Journal of Computer Science 4(6): 421-426, 2008, Science Publications.

[08] IztokSavnik, Peter A. Flach, "Bottom-up Induction of Functional Dependencies from relations", Knowledge Discovery in Databases Workshop WS-93-02, 174-185.

[09] Herbert Schildt, "C The Complete Reference", Fourth Edition, Tata McGraw-Hill Publishing Company Limited, New Delhi, 2000.

[10] Julian M. Scher, CanghuiQiu, "FD-EXPLORER: A pedagogical and Design Tool for Functional Dependency Exploration", in the proceedings of ISECON 2004, v21, 1-7.

[11] Mannila, H., and Raiha K.J., "Algorithms for Inferring Functional Dependencies from relations", Data and Knowledge Engineering, 12(1): 83-99, 1994.

[12] Rajkumar N. Kulkarni and Shivanand M. Handigund, "Abstraction Of Structural Components From Legacy 'C' Program", International Conference on "Advances in Computer Vision and Information Technology (ACVIT – 07)", Aurangabad, India, November 2007, pp. 1523-1530.

[13] Rajkumar N. Kulkarni and Shivanand M. Handigund, "Moulding The Legacy 'C' Programs For Reengineering", International Conference on "Advances in Computer Vision and Information Technology (ACVIT -07)", Aurangabad, India, November, 2007, pp-1531-1537.

[14] Rajkumar N. Kulkarni and Shivanand M. Handigund, "Abstraction of Structural and Behavioral Components from Legacy 'C' Program", 2nd International Conference on Advanced Computing and Communication Technologies (ICACCT 2007), Panipat, Haryana, India, November, 2007, pp 550-554.

[15] Rajkumar N. Kulkarni and Shivanand M. Handigund, "Abstraction Of Structural And Behavioral Components From Legacy 'C' Program", International Journal of Computing Science and Communication Technologies, Vol. 1, No. 1, July 2008, pp 70 – 75.

[16] K KAggarwal and Yogesh Singh, "Software Engineering", Revised second edition, New Age International(P) Limited, 2005, New Delhi.

[17] E. Balaguruswamy, "Programming in ANSI C", third edition, Tata McGraw-Hill Publishing Company Limited, New Delhi, 2006.

[18] T. D. Brown Jr., "C for Basic Programmers", Tata McGraw Hill Publishing Company Limited, New Delhi, 1992.

[19] Dr. Shivanand M. Handigund & Rajkumar N. Kulkarni, "An Ameliorated Methodology for the design of Object Structures from legacy 'C' Program" International Journal of Computer Applications (0975- 8887), Volume 1, No. 13, March 2010, Page No. 61-66.