Generalization of Leaner Object-Oriented Slicing

Rob Law
HTM, The Hong Kong Polytechnic University,
Hung Hom, Kowloon, Hong Kong

Abstract

The Leaner Object-Oriented Slicing technique in [2], albeit effective in providing further code reduction from an object-oriented slice, fails to handle an object-oriented slice with multiple inheritance nets. That is, an object-oriented slice in a tree form with multiple branches is beyond the ability of leaner object-oriented slicing. This paper proposes a new approach which extends the applicability of code reduction by leaner object-oriented slicing. A generalized leaner object-oriented slicing method can now be achieved which provides further code reduction for general object-oriented slices.

Introduction

Given: The following object-oriented program contains a bug and its output is:

```
-6124 toasters were sold
#include <iostream.h>
#include <string.h>
class BillingItem {
protected:
  char name[25];
  int  cost;
public:
  virtual void display() = 0;
};
class Product : public BillingItem {
  int qty_sold;
public:
  Product(char *nm, int qty)
    { qty_sold = qty; strcpy(name, nm); }
  void display() {cout << cost  << ' ' << name << "s were
sold ";}
};
class Service : public BillingItem {
  int manhours;
public:
  Service(char *nm, int mh, int cst)
    { manhours = mh; strcpy(name, nm); cost = cst; }
  void display() { cout << manhours; }
};
class Installation : public Service {
public:
  Installation(char *nm, int hrs, int cst) : Service(nm,hrs,cst) {}
  void display ()
  {cout << "Installed Item: " << name;
   cout << "\nLabour: ";
   Service::display();
   cout << " hours";
   cout << "\nCost: $" << cost << "\n\n"; }
};
main() {
 Product pdsold("toaster", 4);
 pdsold.display();
}
```

Figure 1 A C++ Program

Question: Where is the bug in the program shown in Figure 1?

The solution drawing of the problem in Figure 1 is surely beyond the ability of most existing computer debugging tools, intelligent tutoring systems, and programming environments.

Object-Oriented Program Slicing

An effective fault localization technique was introduced in [2,3] for object-oriented programs. This technique is known as Object-Oriented Program Slicing. An object-oriented slice of an object-oriented program with respect to a class *c* is defined to consist of *c* and all base classes of *c* that could affect (either directly or transitively) the operation of an instance of *c*. In other words, the bug which causes the incorrect operation of an instance of *c* is in this object-oriented slice with respect to *c*. Object-Oriented Program Slicing is simply the procedure used to compute an object-oriented slice.

The C++ code of the object-oriented slice with respect to class *Product* in Figure 1 is shown in Figure 2.

```
#include <iostream.h>
#include <string.h>
class BillingItem {
protected:
  char name[25];
  int cost;
public:
  virtual void display() = 0;
};
class Product : public BillingItem {
  int qty_sold;
public:
  Product(char *nm, int qty)
    { qty_sold = qty; strcpy(name, nm); }
  void display() {cout << cost << ' ' << name << "s were
sold";}
};
main() {
  Product pdsold("toaster", 4);
  pdsold.display();
}
```

Figure 2 An Object-Oriented Slice

The C++ program in Figure 2 returns the same computation as the C++ program in Figure 1 with respect to class *Product*. However, there is 50% reduction in the number of C++ statements that a programmer needs to examine for fault. In other words, a programmer only needs to scrutinize the class definitions of *BillingItem* and *Product* instead of definitions of all classes. This information reduction provides a solid advantage for a programmer to locate bugs. The code reduction percentage would be more significant for large real life object-oriented systems. From their experiments with human subjects, Lyle, as well as Law and Maguire have obtained significant statistical evidence that programmers can locate bugs faster with less amount of code to examine [1,3,4].

Leaner Object-Oriented Slicing

A returned object-oriented slice contains only a subset of code from the original program. Research and development on Object-Oriented Program Slicing still continue. The concept of Leaner Object-Oriented Slicing is an extension of the original approach. A leaner object-oriented slice [LOOS] of an object-oriented slice with respect to a class *c* is defined as a program segment which consists of *c* and all derived classes of *c*. Object-Oriented Slicing is defined as the procedure used to compute a LOOS. The operation of instances of classes in a LOOS can all be affected by *c*. The most feasible application of Leaner Object-Oriented Slicing is the further reduction of irrelevant information from an object-oriented slice.

Generalization of Leaner Object-Oriented Slicing

In this research, a Generalized Leaner Object-Oriented Slice [GLOOS] of an object-oriented slice with respect to a class *c* is defined to consist of the set $LOOS_1$, $LOOS_2$,...., $LOOS_n$ where n is the index of the last inheritance net.

To explain the concept of a GLOOS further, consider the Object-Oriented Slice shown next:

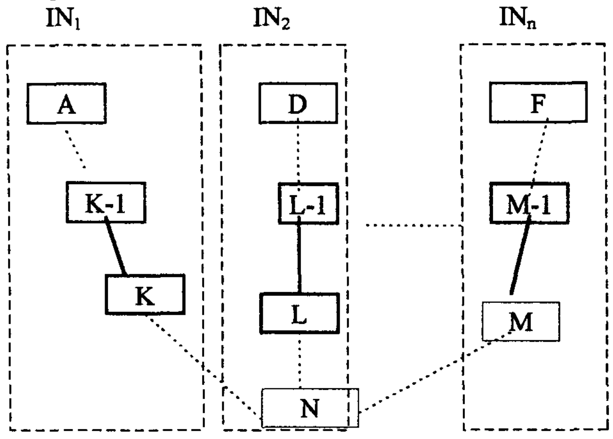

Figure 3 - A Pictorial View of A General Object-Oriented Slice

Figure 3 consists of an object-oriented slice with multiple inheritance nets IN_1, IN_2, ..., IN_n. For IN_1, a user can perform a bottom-up search to look for the occurrence of the first class definition which produces the first incorrect response. Suppose class K in IN_1 is the last one to incorrectly respond, (or class K-1 is the first one to produce a correct response), the search will stop at class K. A LOOS for IN_1 is thus obtained. To obtain a GLOOS, the user only needs to repeat the same procedures for IN_2,....., IN_n.

Source Code of the GLOOS shown in Figure 3 is in Figure 4.

```
#include <iostream.h>
#include <string.h>
class Product : public BillingItem {
 int qty_sold;
public:
 Product(char *nm, int qty)
  { qty_sold = qty; strcpy(name,nm); }
 void display() { cout << cost << ' ' << name << "s were sold";
 }
};
main() {
 Product pdsold("toaster",4);
 pdsold.display();
};
```

Figure 4 A Generalized Leaner Object-Oriented Slice

Conclusions

The way object-oriented programs are developed makes the debugging process much easier as compared to the process of debugging procedural programs. This is because the related operations and the data required by these operations are packed into a class. To locate a bug, the programmer only needs to search for the class which is responsible for the incorrect behaviour. In this research, we have successfully extended the applicability of Leaner Object-Oriented Slicing. A GLOOS can provide further code reduction for general object-oriented slices with single or multiple inheritance nets.

References

[1] Law, R.C.H., 1993, Evaluating the Program Slicing Technique, *SIAST TODAY,* 4(6), p.6.

[2] Law, R.C.H. and Maguire, R.B., 1994, *A University of Regina Object-Oriented Program Slicer*, Technical Report, University of Regina, Canada.

[3] Law, R.C.H. and Maguire, R.B., 1996, Debugging of Object-Oriented Software, *Proceedings of the 1996 Conference on Software Engineering & Knowledge Engineering,* Reno, Nevada, pp. 77-84.

[4] Lyle, J.R., 1984, Evaluating Variations on Program Slicing for Debugging, *Ph.D. Thesis*, University of Maryland.