PROGRAM SLICING BASED ON INTERESTING INDEX

Wu Fangjun* Yi Tong**
*(Dept of Computer Science and Eng., Southeast University, Nanjing 210096)
**(Laboratory of Computer Center, Yichun University, Yichun 336000)

Abstract With the scale of programs becoming increasingly bigger, and the complexity degree higher, how to select program fragments for slicing has become an important research topic. A new type of criterion called interesting index is proposed for selecting parts of procedures or procedure fragments to do program slicing. This new criterion considers not only the subjective aspects in users, namely users' emphasis on the time efficiency, storage capacity or readability, but also the objective aspect in large procedures. It also represents the benefit of the users, while displaying the many-faceted roles that program slicing plays. In this way users can proceed with program slicing to large systems or unfinished systems.

Key words Program analysis; Program slicing; Relative complexity of types; Interesting slicing

I. Introduction

The concept of program slicing proposed by M. Weizer[1], is used to extract statements from source program, which influence the particular variables of the particular statement. Thus, the source program is analyzed through its corresponding program slice.

Ever since the notion of program slicing being put forward, researchers have launched deep and extensive study on it, having obtained many achievements and made it support software engineering activities such as program understanding, testing, debugging, reverse engineering, and maintenance[2-10]. Previous program slicing lay emphasis on complete systems. However, with the development of computer technology, all kinds of software systems have been developed in various domains, with their scale being larger and larger and their complexities higher and higher. Therefore, it is impossible for people to analyze the whole systems. In this case, program slicing to whole systems becomes unnecessary, because it is sufficient for users to take out parts of systems that really interest them to slice. Therefore, the concept of interesting slicing will be introduced here to resolve this problem.

II. Interesting Index

A big software system is usually constituted by a series of small subsystems, which in their turn are constituted by smaller model pieces. Each model sustains certain function, and models pass information via ports. Generally one model scarcely associates with other models, especially in some high cohesion and low couple systems. So it is possible for us to select partial programs to slice. On one hand, in the selection of programs or program fragments we should consider not only the factors of the subjective aspect in users, but also the factors of the objective aspect in large-scale programs. On the other hand, it reflects

1Manuscript received date: November 17, 2003; revised date: December 18, 2003.
not only users' interest, but also the roles program slicing plays in supporting program understanding, testing, debugging, reverse engineering, and maintenance. As a common practice, we can investigate a program from the following four aspects\cite{11}:

1. Correctness

One purpose of program slicing is to find out implicit mistakes. However, a program does not directly connect with slicing whether it has error or not. Therefore we neglect this aspect.

2. Time efficiency

Strictly speaking, execution time of a program means the necessary time when the program runs in computer. It should be the total amount of time that each statement executes in the program. And execution time of a statement equals to the execution numbers of the statement (also call frequency) multiplied by the time per execution of the statement. But, execution time of a statement relies on software and hardware environment, namely instruction system, CPU speed, quality of target code, and the kind of programming language developed. Under the same environment, if the execution time prolongs, it involves more possibility for errors. In these cases, slicing becomes more valuable. Here, we introduce the concept of operation time to reflect the influence that program execution time casts over program slicing. Briefly, we roughly substitute program execution time mainly as execution time of all operations in the program.

Definition 1 Execution time: under some particular environment, time needed to execute an operation \( op \) is called execution time about \( op \), denoted by \( \text{Exectime}(op) \).

Here, operations not only include commonly used arithmetic operation, logic operation, relation operation, assignment operation, etc., but also include location operation, condition operation, pointer operation, comma operation, read operation, write operation, etc. Operation can be unary operations; also can be dyadic operations or multi-item operations.

3. Storage capacity

The storage capacity of a program means the biggest storage capacity during execution of the program. The bigger the storage capacity of the program, the more complex the program, and the more the time needed in correcting error, debugging and maintenance. Therefore, it becomes more necessary to slice. For this reason, we use the concept of storage capacity to reflect the influence of storage capacity over program slicing. Generally, a program contains two parts, which are operators and operands. We can use storage capacity of operands to represent storage capacity of the program approximately.

Definition 2 Storage capacity: under some specific environment, bytes taken by a datum \( v \) (constant or variable) are called storage capacity of this type of data while the program is running, denoted by \( \text{Storage}(v) \). Its value has something to do with the programming language.

Data types in Definition 2 refer to integer data type, float data type, char data type, derived data type, enumeration data type, array data type and string data type, etc. Different programming languages have different data types. Especially, enumeration data type and integer data type occupy the same percentage in storage capacity.

(1) storage capacity of an array data type: an array data type could be supposed as follows \(<\text{type name}><\text{array name}>[<\text{size of array}>] = \{<\text{initial value form}>\}, \) the storage capacity of array datatye is \( \text{arrayname}[\text{size}] = \{\cdots\} \), thus makes following equation workable:

\[
\text{Storage(arrayname)} = \text{size} \times \text{Storage(datatype)};
\]