Function Selection Strategies using Perason Correlation and a Number of Def-use Data-flows between Functions

I. Function Selection Strategy using Pearson Correlation

This function selection strategy is same to that of CONBRIO except that Pearson correlation metric is used to measure dependency between functions, instead of the conditional probability. Suppose that a target program with two functions f and g has n $(= \alpha + \beta + \gamma + \delta)$ system test executions where α, β, γ, and δ are the numbers of the system test executions that execute both f and g, only f, only g, and neither f nor g, respectively. Based on the observations of f and g in the system executions, we compute *Pearson ϕ correlation coefficient* between f and g as follows [1] (higher ϕ coefficient means f and g are more closely related):

$$\phi(f,g) = \frac{\alpha\delta - \beta\gamma}{\sqrt{(\alpha+\beta)(\gamma+\delta)(\alpha+\gamma)(\beta+\delta)}} \tag{1}$$

$\phi(f,g)$ is used as a target function f's dependency on g (and vice versa) instead of the conditional probability $p(g|f)$.

II. Function Selection Strategy using a Number of Def-use Data-flows between Functions

This function selection strategy measures def-use dependency of function `f` to `g` as follows [2] and selects `g`s on which dependency of `f` is higher than the median dependency between all function pairs in a target program:

- For primitive variables used in `f`: it counts a number of def-use instances whose def is in `g`.
- For pointer variables used in `f`: it counts a number of def-use instances whose def is in `g` in a same way for a primitive variable.
- For pointer variable whose dereferenced value (i.e., `*p`) is used in `f`: this heuristic identifies all aliases of `p` (saying `p'`) used in `f` and counts a number of def-use instances whose def is in `g` on all `*p` and `*(p')`.

For example, a number of def-use dependency from `f` to `g` in Figure 1 is 3 (= 1+1+1) as follows:

- A number of def-use instance for `a2` at line 15 is 1 because `a2` is defined by `g(p)`'s return value at line 14.
- A number of def-use instance for `a1.n` at line 15 is 1 because `a1.n` is defined at line 6 of `g` through pointer `p` (see line 14).
- A number of def-use instance for `g1` (a global variable) at line 15 is 1 because `g1` is defined in line 7 of `g`.

```
 1: struct A{
 2:   int n;
 3: };
 4: int g1;
 5: int g(struct A *p2){
 6:   p2->n = 1;
 7:   g1 = 2;
 8:   return g1+1;
 9: }
10: int f(){
11:   struct A a1, *p;
12:   int a2;
13:   p = &a1;
14:   a2 = g(p);
15:   return a2 + a1.n + g1;
16: }
```

Fig. 1. Example of counting def-use dependency

[1] $\phi(f,g)$ ranges from -1 (all system tests execute exclusively either f or g) to +1 (all system tests execute either both f and g or none of them). ϕ coefficient cannot be computed for a function which always executes (e.g., `main`) or never executes with given system test cases. We assign 0 to the correlation with a such function (i.e. no positive or negative correlation).

[2] A structure variable is considered as a set of its primitive field variables (and its nested and linked structure variables through `struct` pointer field variables).