CPTS515 Midterm

Yuzhu Feng

11522898

Preparation: if the input-output (in,out) distribution are the same for integer linear polynomial functions f1 and f2, can f1 and f2 be different?

for two integer linear functions f1 and f2, f1 and f2 must be the same iff the input-output distribution are the same

(Proof by contradiction) Suppose we have two different integer linear polynomial functions f1 and f2

$f_1(x_1,\dots,x_k) = b_1 + \sum_{1\le i\le k} a_{1i}x_i$

$f_2(x_1,\dots,x_k) = b_2 + \sum_{1\le i\le k} a_{2i}x_i$

and $f_1(x_1,\dots,x_k) - f_2(x_1,\dots,x_k) = 0$ for all $(x_1,\dots,x_k)$

Then $f_1(x_1,\dots,x_k) - f_2(x_1,\dots,x_k) = \sum_{1\le i\le k} a_{1i}x_i - \sum_{1\le i\le k} a_{2i}x_i + b_1 - b_2$

$= \sum_{1\le i\le k}(a_{1i} - a_{2i})x_i - (b_1 - b_2)$= 0 for all $(x_1,\dots,x_k)$

Then $a_{1i} - a_{2i} = 0\ for\ all\ 1 \le i \le k$ and $b_1 - b_2 = 0$

Then $a_{1i} = a_{2i}$ for all $1 \le i \le k$ and $b_1 = b_2$

Then $f_1 = f_2$, which is contradictory to the assumption.

Therefore, for two polynomial functions f1 and f2, if the input-output distribution are the same, f1 must be the same as f2.

And it's self-evident that the input-output distribution must be the same if two polynomial functions f1 and f2 are the same, according to the definition of function.

However, for two different functions f1 and f2, there are infinite integer solutions for the equation $f_1(x_1,\dots,x_k) - f_2(x_1,\dots,x_k) = \sum_{1\le i\le k}(a_{1i} - a_{2i})x_i - (b_1 - b_2) = 0$ if $\gcd((a_{11} - a_{21}),\dots,(a_{1k} - a_{2k}))$ divides $b_1 - b_2$ (i.e. $(b_1 - b_2)\ div\ \gcd((a_{11} - a_{21}),\dots,(a_{1k} - a_{2k})) \ne 0)$, according to the property of linear Diophantine equation[1].

Definition 1: f1 and f2 are **functionally equivalent** if f1 and f2 are the same

Definition 2 : f1 and f2 are **functionally similar** if f1 is different from f2, but the equation $f_1(x_1,\dots,x_k) - f_2(x_1,\dots,x_k) = \sum_{1\le i\le k}(a_{1i} - a_{2i})x_i - (b_1 - b_2) = 0$ has infinite integer solutions. Otherwise f1 and f2 are **functionally different**.

Definition: two programs are **behaviorally equivalent** if the conditions below are satisfied:

1. P1 and P2 have the same number of lines of machine code.
2. PC1(Program Counter of P1) and PC2 move in the same way during program execution
3. contents in data memory and registers are the same for each step of execution

Feed the same set of input $\{(x_1, \ldots, x_k) | x_1 \in Z, \ldots, x_k \in Z, k \geq 1\}$ into P1 and P2

    if the output of P1 and P2 are the same:

        then P1 and P2 implement the same function f

Then we'll divide the pairs of programs into four categories:

1. both functionally similar/equivalent(P1 and P2 implement functionally similar functions f1 and f2) and behaviorally similar/equivalent(PC movement is similar and read-write behavior of their data matrix are similar)
2. only functionally similar/equivalent
3. only behaviorally similar
4. neither functionally similar nor behaviorally similar

the similarity distribution is 1. > 2. > 3. > 4. (I prioritize functionality over behavior)
More precisely, the similarity is **First-class-A > First-class-B > First-class-C > First-Class-D > Second-class-A > Second-Class-B > Third-Class-A > Third-Class-B > Fourth-Class**

We'll use the algorithm below that returns δ_p to obtain the behavioral similarity between two programs P1 and P2 in this class, and determine if P1 and P2 are **behaviorally similar**

Define $\delta_C = \frac{number\ of\ same\ address\ for\ PC1\ and\ PC2\ at\ the\ same\ time\ during\ program\ execution}{\#(PC1)+\#(PC2)}$ to denote the similarity of program execution between P1 and P2

, where #(PC1) denotes the total number of times of PC1's move during P1's execution, and #(PC2) denotes the total number of times of PC2's move during P2's execution

Define $\delta_M = \frac{average(number\ of\ same\ register\ content)}{number\ of\ registers\ in\ P1 + number\ of\ registers\ in\ P2} + \frac{average(number\ of\ same\ matrix\ content)}{k1*k1+k2*k2}$,

where

average(number of same register content)

$$= \frac{\sum_i^{\max(number\ of\ execution\ for\ P1, number\ of\ executio\ \ for\ P2)} number\ of\ same\ register\ content\ for\ ith\ execution}{number\ of\ execution\ for\ P1 + number\ of\ execution\ for\ P2}$$

number of execution for P1 is the number of times PC1 has traversed during P1's execution

number of execution for P2 is the number of times PC2 has traversed during P2's execution.

δ_M denotes the similarity of memory content(including register content) between P1 and P2

Define $\delta_P = \delta_C + \delta_M$ to denote the similarity between P1 and P2 that both implement the same function f

Define a threshold δ_t (a real number between 0.0 and 1.0 defined by engineers), say 0.75, to classify programs as 1. **behaviorally similar** if $\delta_p \geq \delta_t$ 2. **behaviorally different**(complementary/opposite to behaviorally similar) if $\delta_p < \delta_t$

Thus if $\delta_p \geq \delta_t$,the corresponding function pair P1 and P2 belong to class one or class three, otherwise they belong to class two or class four.

First class: for P1 and P2 that are both at least functionally similar and behaviorally similar

Observation: it's false that P1 is the same as P2 if P1 and P2 are behaviorally equivalent.

Proof: (Proof by contradiction)
Suppose P1 and P2 are the same when they are behaviorally equivalent,
Suppose P1 has only single line of code: mov \$t0,\$s1 when \$s1 = 10, and \$t0 = 0 before the execution. P2 also has only one line of code: add \$t0, \$s1,\$s2 when \$s1 = 10, \$s2 = 0 and \$t0 = 0 before execution. And all the other register contents and data memory content are the same before execution
Then in this case $\mathrm{P1} \neq \mathrm{P2}$, which is contradictory to our assumption
Thus the above three conditions cannot guarantee that P1 and P2 are the same.

But they are definitely very similar because they implement the same function and their register content and memory content are the same during execution.

Subclasses for first class

pairs of programs inside this class that are **both functionally equivalent and behaviorally equivalent** are more similar than any other pairs of programs, and we call those pairs of programs **First-class-A**.

pairs of programs inside this class that are **functionally equivalent and behaviorally similar** are less similar than First-class-A pairs but more similar than any other pairs, and we call such pairs of programs **First-class-B**

pairs of programs inside this class that are **functionally similar and behaviorally equivalent** are less similar than First-class-A and First-class-B but more similar than any other pairs of programs, and we call them **First-class-C**

pairs of programs inside this class that are **functionally similar and behaviorally similar** are least similar in this class(of course they are more similar than lower classes), and we call them **First-class-D**.

Class Two: for P1 and P2 that only functionally similar/equivalent (but behaviorally different)

All the functions in this class are less similar than any function in the first class.
δ_p in this class of program pairs P1 and P2 is always smaller than δ_p in the first class.

Subclasses for second class

Using similar approach in the first class, we name the pairs of programs P1 and P2 as **Second-Class-A** if they are functionally equivalent (but behaviorally different), and as **Second-Class-B** if they are functionally similar (but behaviorally different).

Class Three: for P1 and P2 that are only behaviorally similar/equivalent (but functionally different)

δ_p in this class of program pairs P1 and P2 can be larger than δ_p in the second class, and can be as large as δ_p in the first class.

Subclasses for third class

Using similar approach in the second class, we name the pairs of programs P1 and P2 as **Third-Class-A** if they are behaviorally equivalent (but functionally different), and as **Third-Class-B** if they are behaviorally similar (but functionally different).

And all the pairs in Third-Class-A are more similar than those in Third-Class-B

Class Four: for P1 and P2 that are neither behaviorally similar nor behaviorally similar

δ_p in this class of program pairs P1 and P2 is always smaller than δ_p in the first and third class, but may be as large/small as δ_p in the second class(because all the programs in the second class are only functionally similar). We'll use a more accurate way to measure the similarity of two programs P1 and P2 in this class.

All the pairs in this class are called Fourth-Class

Algorithm:

The algorithm servers as a binary classifier:
Input: two black-box programs P1 and P2

```
Check whether P1 and P2 are functionally equivalent or functionally similar:
    feed a large number of integer tuples (for example 10,000) $(x_1, \dots, x_k)$ , $k \geq 1$ from the set
$\{(x_1, \dots, x_k) \mid x_1 \in Z, \dots, x_k \in Z, k \geq 1\}$ into both program P1 and P2 to get the output $y_{1i}, y_{2i}, 1 \leq \mathrm{i} \leq \mathrm{k}, \mathrm{k} \geq 1$
    if all the output are the same for each input, i.e. if $y_{1i} = y_{2i}$ , $1 \leq i \leq k, k \geq 1$:
        P1 and P2 are functionally equivalent
        return "functionally equivalent"
    if most of the output (for example 9900) are the same for all the input:
        P1 and P2 are functionally similar (but not functionally equivalent)
        return "functionally similar"
    return false // P1 and P2 are not functionally similar/P1 and P2 are functionally different

Check whether P1 and P2 are behaviorally similar:
    Check whether P1 and P2 are behaviorally equivalent using the three conditions:
        if all the three conditions are satisfied: //P1 and P2 are behaviorally equivalent
            then return "behaviorally equivalent"
    compute $\delta_p$ for P1 and P2
    if $\delta_p \geq \delta_t$:
        then P1 and P2 are behaviorally similar
        return "behaviorally similar"
    else:
        return false
Classify the pair P1 and P2 using the four classes
return $(\delta_p, \text{the class the pair belong to})$ // for example, return (0.78,First-Class-A)
```

The returned pair is the metric for evaluating the similarity between two black-box programs P1 and P2.

If, for example, the returned $(\delta_p, Class)$ shows that P1 and P2 belong to the first class and they are both functionally and behaviorally equivalent, then they are very similar to each other; if P1 and P2 belong to the second class then they are definitely less similar than any pair of black-box programs P3 and P4 that belong to the first class, and so forth.

Question: How to compare two pairs of programs (P1,P2) and (P1,P3) if they are in the same class, for example how to compare (P1,P2) and (P1,P3) if they both belong to First-Class-A?

Answer: Compare the δ_p value of the two pairs, the pair of programs that has greater δ_p value is more similar.

Reference

[1] *Linear Diophantine Equations*, gauss.math.luc.edu/greicius/Math201/Fall2012/Lectures/linear-diophantine.article.pdf.