Project report for Programming Languages:
Automated Verification Of Correct Answers

Meghana Kotagiri, IMT2014034
G Neha, IMT2014018

Mentored by: Prof. Sujit Kumar Chakrabarti, Prof. Manish Gupta

Abstract

MOOCs and online learning platforms are changing the way we impart education and gain knowledge. The courses offered, especially those related to programming, require students to submit code for a problem. These institutions have to somehow check these solutions and provide appropriate feedback to students to facilitate learning. As the number of students enrolled for such courses is huge, manual checking is not practically feasible. Not only MOOCs, even universities have to conduct numerous programming assignments, which makes correction and feedback of these assignments a great challenge. In this report, we explore how to automate this whole process, i.e, identifying right submissions and grading the incorrect ones appropriately. The proposed approach and our contribution to it are presented in this report.

Contents

1 Understanding The Problem

As stated earlier, our primary objective is: Given a set of solutions submitted by students (denote by S), identify the correct ones and grade the submissions appropriately. We can decompose our problem into the following subproblems:

Step 1 Given a set of submitted programs and a correct solution, identify the correct solutions from submitted answers (submissions i.e. S). Test cases that can discriminate wrong submissions from the correct ones need to be generated. And then all submissions need to be tested against these test cases to find out which are the correct submissions (denote by $S_{correct}$) and the incorrect ones (denote by $S_{incorrect}$).

Step 2 As every student has her own programming style, S might contain different approaches to solve the same question. Given the correct solutions ($S_{correct}$), find what are the unique approaches used by students to tackle the question correctly. We will call these unique approaches as GOLD STANDARDS ($S_{gold-standard}$).

Given $S_{gold-standard}$, and a submission from $S_{incorrect}$, find out which approach student was trying to follow. We require to find a tuple (P, G) where $P \in S_{incorrect}$ and $G \in S_{gold-standard}$ is the approach that P is closest to.

Step 3 Given a pair of submitted solution and the corresponding gold standard (P, G), find the least number of steps to convert G to P using program repair techniques and assign appropriate scores.

We have worked on Step 2 and our proposed solution is based on identifying similar programs by comparing their Program Dependence Graphs (PDG). It is motivated by GPLAG [1], whose key contents (valid in our context) are discussed in subsections 1.1 and 1.2.

1.1 Key Concepts and Definitions

The core idea is based on the concept of PDGs. A PDG is graphical representation of program, where statement types are represented as graph nodes, and edges represent control & data dependency between the nodes. Intutively, PDG capture the program logic. Formally it can be defined as follow:

Definition 1. The *program dependence graph* G for a procedure P is a 4-tuple element $G = (V, E, \mu, \delta)$, where

- V is the set of program vertices in P
- $E \subseteq V \times V$ is the set of dependency edges, and $|G| = |V|$
- $\mu : V \to S$ is a function assigning types to program vertices
- $\delta : E \to T$ is a function assigning dependency types, either data or control, to edges

Definition 2. A bijective function $f : V \to V'$ is a *graph isomorphism* from a graph $G = (V, E, \mu, \delta)$ to a graph $G' = (V', E', \mu', \delta')$ if

- $\mu(v) = \mu'(f(v))$
- $\forall e = (v1, v2) \in E, \exists e' = (f(v1), f(v2)) \in E'$ such that $\delta(e) = \delta(e')$
- $\forall e = (v1', v2') \in E', \exists e = (f^{-1}(v1'), f^{-1}(v2')) \in E$ such that $\delta(e') = \delta(e)$

Definition 3. An injective function $f : V \to V'$ is a *subgraph isomorphism* from G to G', if there exists a subgraph $S \subset G'$ such that f is a graph isomorphism from G to S.

Definition 4. A graph G is *γ-isomorphic* to G' if there exists a subgraph $S \subseteq G$ such that S is subgraph isomorphic to G' , and $|S| \geq \gamma|G|$, $\gamma \in (0, 1]$

1.2 Invariance of PDGs

As long as the *logic encoded* in the two programs is same, the code in both can vary in some specific ways. We briefly discuss below these ways, along with why these alterations have no effect on PDGs of programs.

1. Format Alteration: Inserting and removing blank statements/ comments.

   Trivailly, format alteration has no dependence on PDG of a program.

2. Identifier Renaming: Identifier(variables, function) names are changed without violating program correctness.

   As PDG concerns itself with the type of identifier and not its name, this change has no effect on PDG.

3. Statement Reordering: Program statements are reordered without causing errors and affecting sequential dependencies.

To keep the functionality of the program intact, one can reorder program statements only if they are not dependent on one other. Therefore, dependence edges still remain te same, inturn keeping the PDG same.

4. Control Replacement: Replacing *while* with *for*, changing *if* conditions to their negations.

   As both *while* and *for* have node type *Loop*, such changes do not significantly alter the program PDG.

5. Code Insertion: Immaterial code insertion which doesn't affect original program logic.

   Extra code results in new nodes and edges, but it does not interfere with the exististing relationships and structure. Therefore, the original PDG will still be subgraph isomophic to new PDG with code insertion.

Therefore, we can make the following claim:

Claim 1. *Given G_1, G_2 are PDGs of program P_1, P_2 respectively. If G_1 is $\gamma -$ isomorphic to G_2, then we say that logic encoded in P_1 and P_2 is same.*

2 Approach Outline

Given Gold Standards ($S_{gold-standard}$), find the approach that a submitted solution ($P_{test} \in S$) is closest to.

2.1 Steps to be followed

1. Create PDGs of programs in $S_{gold-standard}$ and P_{test}.

2. Parse the PDGs as graphs appropriately.

3. For each $G_{gold-standard} \in S_{gold-standard}$,

   do following on $(G_{gold-standard}, G_{test})$ pair:

4. Perform space reduction using filters.

5. Pass the filtered graphs to the gamma isomorphism algorithm

6. Check for gamma isomorphism. If true then output G_{test} is similar to $G_{gold-standard}$

2.2 Pseudo Code

The following code takes two programs (P,P') as inputs and finds out if the given programs are $\gamma - isomorphic$ or not.

Algorithm 1 PDG comparison Algorithm

```
1: procedure COMPARE(P, P')
2:     PDG ← generatePDG(P)
3:     PDG' ← generatePDG(P')
4:     (G, G') ← (parse(PDG), parse(PDG'))
5:     if !$Filter(G, G')$ then return false
6:     else
7:         if GammaIsomorphic($G, G'$) then return true
8:         else return false
```

2.3 Code explanation[1]

We have chosen to use C-program submissions as test cases for our project. The program dependence graphs for the C-programs were generated using the *frama-c* tool. The dot file generated by frama-C was parsed to create the graph data structure.

The structure of the dot file was exploited to create graph with labelled edges and labelled nodes. *CreateGraph.java* file parses the dot file and creates the graph. The *GraphInfo.java* file contains the graph data structure which utilizes *jgrapht* library.

The Graph isomorphism algorithm takes as input the dot file of the correct solution(gold-standard) and dot file of the student submission. After constructing the program dependence graphs, these graphs are subjected to filters before passing them as inputs to the gamma isomorphism function. These filters reduce the search space. Filter-1 is based on the size of the graphs generated. Any graph of size less than the threshold fixed would be ignored. Filter-2 is based on the size criterion for gamma isomorphism. If both the graphs manage to pass through these filters, then they will checked for gamma isomorphism.

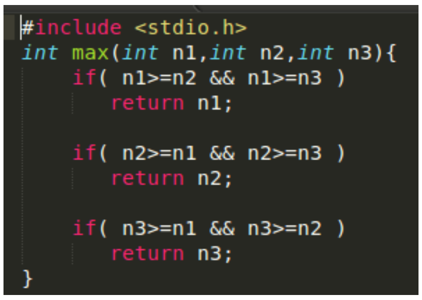
Gamma Isomorphism Algorithm:

[1]Please refer to code files for better understanding.

Check the PDG of the original solution with the PDG of the student submission for γ- isomorphism. If the algorithm returns true then both of them are similar. If it fails, create a set 'S' containing all the vertices of the PDG of original solution. Generate all possible subsets of this set of the size equal to $(|S| - 1)$. A Subgraph is formed from the subset and checked with student solution for subgraph isomorphism. This is repeated for all the subsets. Then we decrease the size and generate subsets of the new size. Generate subgraphs and check for subgraph isomorphism. While checking for subgraph isomorphism, if it outputs true then we return true and terminate the algorithm. Else, if it outputs false, we continue checking for subgraph isomorphism till the gamma size criterion fails. This logic has been coded in the *Graph_isomorphism_check1.java* file.

3 Results

Consider the following test files:

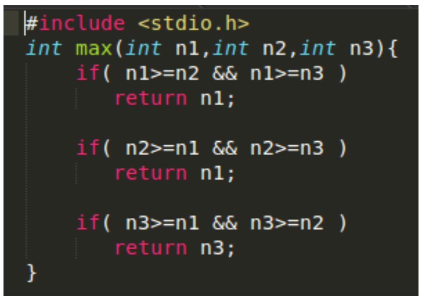


```
#include <stdio.h>
int max(int n1,int n2,int n3){
    if( n1>=n2 && n1>=n3 )
        return n1;

    if( n2>=n1 && n2>=n3 )
        return n2;

    if( n3>=n1 && n3>=n2 )
        return n3;
}
```

(a) max1 : correct solution



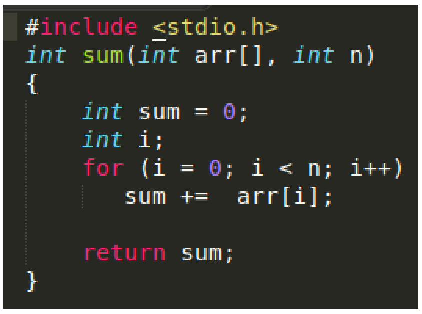
```
#include <stdio.h>
int max(int n1,int n2,int n3){
    if( n1>=n2 && n1>=n3 )
        return n1;

    if( n2>=n1 && n2>=n3 )
        return n1;

    if( n3>=n1 && n3>=n2 )
        return n3;
}
```

(b) max2 : incorrect solution

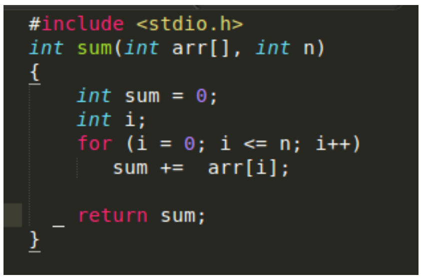
Figure 1: Function to compute max of 3 numbers



```
#include <stdio.h>
int sum(int arr[], int n)
{
    int sum = 0;
    int i;
    for (i = 0; i < n; i++)
        sum +=  arr[i];

    return sum;
}
```

(a) sum1 : correct solution



```
#include <stdio.h>
int sum(int arr[], int n)
{
    int sum = 0;
    int i;
    for (i = 0; i <= n; i++)
        sum +=  arr[i];

    return sum;
}
```

(b) sum2 : incorrect solution

Figure 2: Function to compute sum of an array

As it can be observed, code 1a and 1b differ slightly. The student has returned the value($n1$) instead of value($n2$) for one of the conditions.

In case of calculating sum of an array, 2b is almost correct except the bound checking which should have been $i < n$.

In both the cases, student has almost captured the logic to get the solution in both the cases.

Results of $\gamma - isomorphism$ algorithm on above test cases:

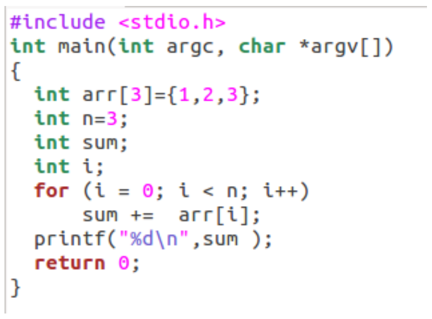
Programs	**Passes Filter**	**Are** $\gamma - Isomorphic$
max1.c & max2.c	yes	true
sum1.c & sum2.c	yes	true
max1.c & sum1.c	no	false

4 Limitations

1. For the size filter, the size criterion varies according to the programs. Based on the size of the correct solutions, this size has to be decided and fixed accordingly.

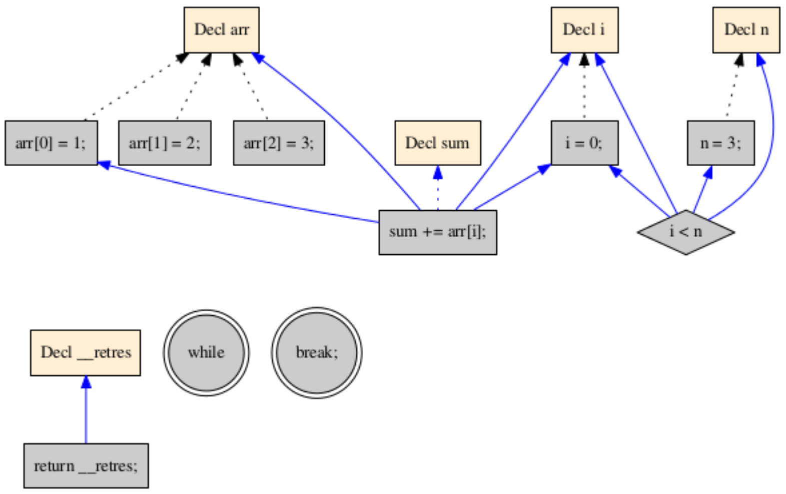
2. The output generated by frama-c is not consistent. For certain files, all the nodes were not generated. This could result in extra computations and wrong results while checking for gamma isomorphism.

   For example,consider the following code:



```
#include <stdio.h>
int main(int argc, char *argv[])
{
  int arr[3]={1,2,3};
  int n=3;
  int sum;
  int i;
  for (i = 0; i < n; i++)
      sum +=  arr[i];
  printf("%d\n",sum );
  return 0;
}
```

   For this graph the i++ node is not generated.



Here, we can clearly see that all the nodes are not generated.

3. Finding a suitable value of gamma to check for gamma-isomorphism. The gamma constraint is a lower bound on the size while checking for subgraph isomorphism. While checking for gamma isomorphism, for certain set of solutions, we might need to relax the gamma constraint further. We can learn the value of gamma to be used by running this algorithm for different set of solutions.

4. Algorithm complexity: Both subgraph isomorphism and subset generation algorithms have exponential complexity. For larger graphs, this could result in an increase in the computation time.

5. While identifying unique solutions among student submissions, we are assuming that student submissions contain all possible unique approaches. This assumption could be misleading while comparing student submissions with the set of unique solutions identified. As the student submission could contain a solution approach that might not be identified while checking for unique solutions.

6. Gamma isomorphism operation is not commutative.

   Consider two graphs A of size(no. of nodes) 26 and B graph of size 18. Assume that they both have subgraphs of size 15 that are isomorphic.

Now, consider the gamma filter condition:

$$\text{Filter(G1,G2)} = |G2| \geq (gamma * |G1|)$$

When gamma =0.8, the pair (B,A) would pass the filter. But, the pair (A,B) wouldn't pass the filter.

References

[1] Chao Liu, Chen Chen, Jiawei Han, and Philip S. Yu. 2006. GPLAG: detection of software plagiarism by program dependence graph analysis. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '06). ACM, New York, NY, USA, 872-881. DOI=http://dx.doi.org/10.1145/1150402.1150522