Mutation testing

Cây AST

Trong ngành khoa học máy tính, cây cú pháp trừu tượng (AST, abstract syntax tree) là một cây có giới hạn, có nhãn và có định hướng. Đây là cấu trúc cây mà các nút gốc của cây được gán nhãn bằng các toán tử và các nút là của cây là các toán hạng.[1] Tuy vậy, các lá cũng có khi là các giá trị NULL hoặc là các biến hoặc các hàng. Trong các công đoạn của chương trình dịch, cây AST này được dùng trong bộ phân tích cú pháp như là một trung gian giữa cây phân tích cú pháp (concrete syntax tree) và cấu trúc dữ liệu. Cây cú pháp trừu tượng khác với cây phân tích cú pháp là ở chỗ nó không chỉ quan tâm đến cú pháp mà còn quan tâm đến ngữ nghĩa của chương trình.

A screenshot of a computer

Description automatically generated with medium confidence

Cây cú pháp trừu tượng với đoạn mã của [Giải thuật Euclid](https://vi.wikipedia.org/wiki/Gi%E1%BA%A3i_thu%E1%BA%ADt_Euclid" \o "Giải thuật Euclid):

**while** b ≠ 0

**if** a > b

a:= a − b

**else**

b:= b − a

**return** a

**Mutation testing**

Mutation testing là 1 loại kiểm thử phần mềm, nơi mà chúng ta thay đổi câu lệnh trong source code và check xem test case có thể tìm thấy lỗi hay không.

* Là một kiểu white box testing được sử dụng chủ yếu trong unit testing. Những thay đổi này là những thay đổi nhỏ để đảm bảo không ảnh hưởng tới tổng thể hệ thống.
* Mutation testing dùng để kiểm tra hiệu năng và độ chính xác của chương trình test. Phương pháp này giúp kiểm tra những thiếu sót của chương trình trong khi test. Nó cũng giúp cho việc ước lượng và hoàn thiện bộ test case.

mutpy

MutPy là một công cụ kiểm tra đột biến cho mã nguồn Python 3.3+. MutPy hỗ trợ unittest, tạo báo cáo YAML / HTML và có đầu ra được tô màu. Nó áp dụng đột biến trên cây AST.

Cách mutpy sinh đột biến bậc 1

Mutpy sinh đột biến bậc 1 bằng cách đột biến các nođe có thể đột biến trên cây AST, bằng cách thay thế nó bằng một node khác có cùng tính chất, hoặc thêm một node bao phủ node đó hoặc xoá một node và đưa node con nó bao phủ lên thay thế node đó.

Sinh data bậc 2

Lý do: các HOM Strategy có sẵn của mutpy chỉ sinh ra được rất ít mutation bậc 2, khoảng ½ số mutation bậc 1 do khi sinh chỉ dùng mỗi mutation bậc 1 một lần duy nhất

Giải pháp: tự viết HOM Strategy ALL\_PAIR để tạo ra mutation bậc 2 bằng cách ghép 2 mutation bậc 1 có thể ghép được với nhau. Như vậy số mutation bậc 2 có thể tạo ra khoảng với n là số mutant bậc 1

class AllPairHOMStrategy(HOMStrategy):  
 name = 'ALL\_PAIR'  
  
 def \_\_init\_\_(self, \*args, shuffler=random.shuffle, \*\*kwargs):  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
 self.shuffler = shuffler  
  
 def generate(self, mutations):  
 size\_of\_mutations = len(mutations)  
 for i in range(size\_of\_mutations):  
 first\_mutations = []  
 first\_mutations.append(mutations[i])  
 available\_mutations = []  
 for j in range(i + 1, size\_of\_mutations):  
 available\_mutations.append(mutations[j])  
 self.remove\_bad\_mutations(first\_mutations, available\_mutations)  
 for mutation in available\_mutations:  
 mutations\_to\_apply = []  
 mutations\_to\_apply.append(mutations[i])  
 mutations\_to\_apply.append(mutation)  
 yield mutations\_to\_apply

đoạn code sinh đột biến này được chèn vào file controller.py của mutpy

Sinh data

Sử dụng 1 số thư viện chuẩn của cpython và mutpy để sinh data

Các thuộc tính khai thác

1) typeStatement: loại câu lệnh bị đột biến. Trích xuất thuộc tính này dựa vào node bị đột biến hay các node cha của node bị đột biến.

2) typeOperator, toán tử sử dụng để đột biến. Trong mutpy có 20 toán tử được sử dụng của đột biến là:

* AOD - arithmetic operator deletion
* AOR - arithmetic operator replacement
* ASR - assignment operator replacement
* BCR - break continue replacement
* COD - conditional operator deletion
* COI - conditional operator insertion
* CRP - constant replacement
* DDL - decorator deletion
* EHD - exception handler deletion
* EXS - exception swallowing
* IHD - hiding variable deletion
* IOD - overriding method deletion
* IOP - overridden method calling position change
* LCR - logical connector replacement
* LOD - logical operator deletion
* LOR - logical operator replacement
* ROR - relational operator replacement
* SCD - super calling deletion
* SCI - super calling insert
* SIR - slice index remove

Tên của toán tử có thể trích xuất tử mutation.operator trong file controller.py, sau dòng 338 for mutation in mutations\_to\_apply:

3) typeReturn, kiểu trả về của toán tử bị đột biến. trong môt số trường hợp có thể xác định rõ kiểu trả về của toán tử của đột biến như toán tử logic trả về kiểu bool, toán tử / trả về kiểu float, … lấy tên toán tử bằng cách lấy mutation.node

5) depthOnAST: Độ sâu của mutant trên cây AST. trước khi thực hiện đột biến ta thêm thuộc tính depth cho các node trên cây ast, khi sinh mutant ta có thể dựa vào mutation.node để lấy thuộc tính này luôn

4) distanceBetweenTwoMutant: Khoảng cách giữa 2 node bị đột biến trên cây ast, sử dụng lca để tính

6) result: trạng thái của đột biến khi chạy bộ test, các trạng thái là killed, survived, incompetent, timeout

Các thuộc tính 1, 2, 3, 4 được nhân đôi để biểu diễn 2 vị trí bị đột biến của của một đột biến

Scikit-learn là gì?

Scikit-learn (Sklearn) là thư viện mạnh mẽ nhất dành cho các thuật toán học máy được viết trên ngôn ngữ Python. Thư viện cung cấp một tập các công cụ xử lý các bài toán machine learning và statistical modeling gồm: classification, regression, clustering, và dimensionality reduction.

Thư viện được cấp phép bản quyền chuẩn FreeBSD và chạy được trên nhiều nền tảng Linux. Scikit-learn được sử dụng như một tài liệu để học tập.

LogisticRegression

<https://machinelearningcoban.com/2017/01/27/logisticregression/>

RandomForestClassifier

https://en.wikipedia.org/wiki/Random\_forest

Training

Sử dụng 2 thuật toán LogisticRegression và RandomForestClassifier để training

Sử dụng các tham số mạc định của sklearn để training

Các feature typeStatement, typeOperator, typeReturn sinh ra nhiều binary sub-feature, mỗi sub-feature thể hiện xem thuộc tính đấy có phải là thuộc tính của đột biến ko

1. Em chạy thực nghiệm trên project nào? Số lượng dòng lệnh và số lượng testcases của nó là bao nhiêu? Và sinh ra bao nhiêu đột biến bậc 1 và bao nhiêu đột biến bậc 2?

Em chạy thực nghiệm ở Project Cpython phiên bản 3.7, số dòng lệnh và số đột biến em sẽ để trong file report. Link project https://github.com/python/cpython/tree/3.7

2. Dữ liệu dùng để training và Test mình lấy từ đâu?

Dữ liệu training và test em sinh bằng cách sử dụng mutpy và HOMStrategy ALL\_PAIR tự viết để sinh đột biến, em code thêm 1 số đoạn code để trích xuất thông tin và lưu ra file csv. Em chạy trên khoảng 10 thư viện trong mục Lib.

3. Em có số liệu các đột biến "trùng nhau" ở tập đột biến được "dự báo bị diệt" và tập đột biến bị diệt thực tế là bao nhiêu không?

Em sẽ trích xuất và lưu trong file report.

1. Các feature typeStatement, typeOperator, typeReturn sinh ra nhiều binary sub-feature, mỗi sub-feature thể hiện xem thuộc tính đấy có phải là thuộc tính của đột biến ko.

Đây là các thuộc tính của đột biến ạ, tức là 1 đột biến có kiểu dữ liệu trả về là bool thì cột bool của đột biến này có giá trị là 1, còn các cột int, float, unknown có giá trị là 0.

2. Sử dụng các tham số mặc định của sklearn để training ? Là các tham số nào em?  
tức là em gọi thư viện mạc định của sklearn , chỉ truyền data vào để training chứ không truyền thêm một cái gì khác