TRƯỜNG ĐẠI HỌC CÔNG NGHỆ - ĐHQGHN

**KHOA CÔNG NGHỆ THÔNG TIN**

CÔNG TRÌNH DỰ THI

GIẢI THƯỞNG “SINH VIÊN NGHIÊN CỨU KHOA HỌC”

NĂM 2017

Tên công trình: **Xây dựng công cụ kiểm chứng một số tính chất của chương trình dựa trên kĩ thuật thực thi tượng trưng**

Họ và tên sinh viên:

**Nguyễn Đức Thuần** Nam Lớp: K59CLC

**Nguyễn Bá Tú**  Nam Lớp: K59CLC

Khoa: Công nghệ thông tin

Người hướng dẫn: TS. Tô Văn Khánh

Mục lục

[Mở đầu 4](#_Toc476899493)

[I. Giới thiệu 4](#_Toc476899494)

[II. Thực thi tượng trưng 4](#_Toc476899495)

[1. Tổng quan về kĩ thuật thực thi tượng trưng 4](#_Toc476899496)

[2. ứng dụng của SE 4](#_Toc476899497)

[3. Các công cụ liên quan 4](#_Toc476899498)

[3.1 KLEE 4](#_Toc476899499)

[3.1.1. Tổng quan (Abstract) 4](#_Toc476899500)

[3.1.2. Mô hình hoạt động của KLEE 4](#_Toc476899501)

[3.1.3. Chức năng 5](#_Toc476899502)

[3.1.4. Hạn chế 5](#_Toc476899503)

[3.2 Cascade 6](#_Toc476899504)

[3.2.1 Tổng quan 6](#_Toc476899505)

[3.2.2 Thiết kế hệ thống của Cascade 6](#_Toc476899506)

[3.2.3 Mô hình bộ nhớ 7](#_Toc476899507)

[3.3 Một số công cụ khác 8](#_Toc476899508)

[**III.** **Công cụ kiếm chứng chương trình** 9](#_Toc476899509)

[**1.** **Trừu tượng hóa chương trình dựa trên thực thi tượng trưng** 9](#_Toc476899510)

[**Cây cú pháp trừu tượng (AST)** 10](#_Toc476899511)

[**Xây dựng đồ thị luồng điều khiển** 12](#_Toc476899512)

[**Đánh chỉ số (indexing) cho các tham biến và xây dựng biểu thức metaSMT** 13](#_Toc476899513)

[**Công thức tổng quát cho phương thức** 14](#_Toc476899514)

[**Kết hợp điều kiện của người dùng và đưa vào bộ giải** 14](#_Toc476899515)

[**2.** **Framework** 15](#_Toc476899516)

[**3.** **Case Study** 16](#_Toc476899517)

[IV. Thực nghiệm 19](#_Toc476899518)

[1. Công cụ 19](#_Toc476899519)

[1.1 Tạo cây trường tượng (Abstract Syntax Tree) từ mã nguồn 19](#_Toc476899520)

[1.2 Tạo đồ thị luồng điều khiển (Control Flow Graph – CFG) từ cây cú pháp trừu tượng 19](#_Toc476899521)

[1.3 Unfold cfg 19](#_Toc476899522)

[1.4 Đánh chỉ số (indexing) cho các biến và tham số trong phương thức (metaSMT) 19](#_Toc476899523)

[1.5 Đưa ra công thức tổng quát 19](#_Toc476899524)

[1.6 Đưa vào bộ giải và nhận kết quả 19](#_Toc476899525)

[2. Một số ứng dụng thực nghiệm 19](#_Toc476899526)

# Mở đầu

# Giới thiệu

Tổng quan về kiếm chứng chương trình

# Thực thi tượng trưng

## Tổng quan về kĩ thuật thực thi tượng trưng

## Ứng dụng của SE

## Các công cụ liên quan

### 3.1 KLEE

#### 3.1.1. Tổng quan (Abstract)

KLEE là một công cụ thực thi tượng trưng, có khả năng sinh các ca kiểm thử một cách tự động mà đạt được độ bao phủ cao trên một tập đa dạng của các chương trình phức tạp và môi trường chuyên sâu (được dịch từ “KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs” - <http://hci.stanford.edu/cstr/reports/2008-03.pdf> ). Klee thuộc nhóm công cụ sử dụng kĩ thuật thực thi tượng trưng động ( Dynamic Symbolic Execution) với việc sử dụng kết hợp các thực thi tượng trưng và thực thi cụ thể

#### 3.1.2. Mô hình hoạt động của KLEE

metaSMT

Boolector

Z3

STP

Constraint  
Independence

Branch Cache

Counterexample  
Cache

Hình 1: Sơ đồ hoạt động của KLEE

* Constaint Independence

Tại bước này, Klee loại bỏ các ràng buộc không cần thiết. Ví dụ ta có ràng buộc C + {x + 2y = 5; y > 3; z > 20 ; w > 5} khi đi xuống một nhánh E = x > y thì ta có thể loại bỏ z > 20 và w > 5 mà không anh hưởng đến E.

* Branch Cache

Klee lưu lại kết quả truy vấn đến các nhánh vào bộ đệm branch cache

* Counterexample Cache

Klee thiết lập một tập hợp các điều kiện hoặc thiết lập một Counterexample Cache (phản ví dụ).

Klee sử dụng counterexample Cache để có thể đưa ra được kết quả một cách nhanh hơn để tiết kiệm thời gian cũng như chi phí.

* Tiếp theo, các ràng buộc được đưa đến các bộ giải ràng buộc (Constrain solver) thông qua cơ chế mạng metaSMT. Klee sẽ lựa chọn bộ giải phù hợp nhất cho từng trường hợp.

#### 3.1.3. Chức năng

Chức năng chính của KLEE là sinh ra các ca kiểm thử một cách tự động với độ phủ cao, các ca kiểm thử có khả năng phát hiện được lỗi của chương trình thông qua các tập tin đầu ra kết quả.

Ngoài ra, KLEE còn được sử dụng trong các công cụ khác với những ứng dụng đa dạng khác nhau, các công cụ này đều thừa kế phát triển hay mở rộng KLEE dựa trên khả năng thăm dò các đường thi hành của chương trình và tự động sinh các ca kiểm thử. Ví dụ:

* EDS – Execution Synthesis: công cụ có khả năng sửa lỗi chương trình C một cách tự động
* KLEENET: có khả năng tìm ra những lỗ hổng bảo mật trong hệ thống phân phối trước khi triển khai và tạo điều kiện cho việc mở rộng hệ thống mạng
* KLOVER: Có khả năng sinh ca kiểm thử tự động cho chương trình C++

#### 3.1.4. Hạn chế

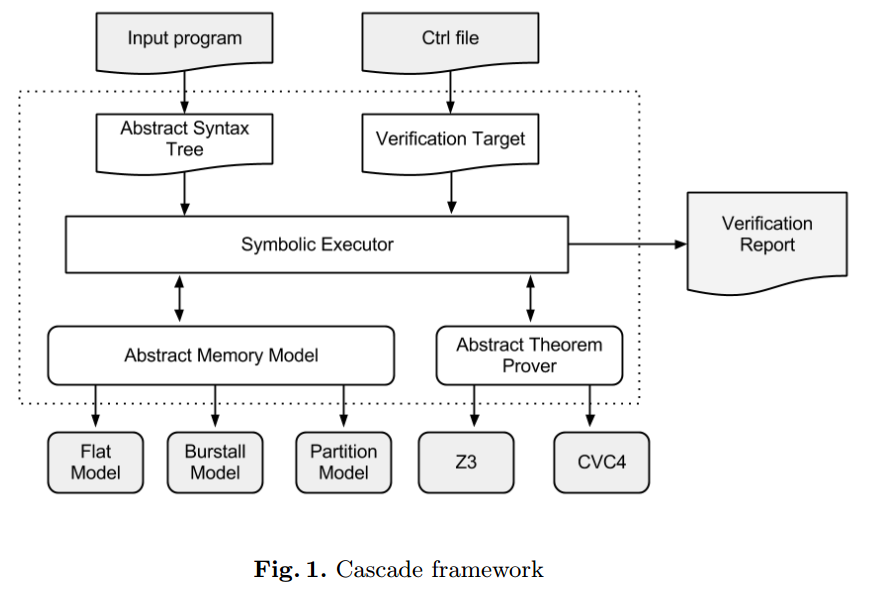
* KLEE chưa xử lý được vấn đề bùng nổ đường thi hành ( The path explosion problem )
* KLEE cũng không thể thực hiện với lặp với điều kiện tượng trưng và các đối tượng có kích cỡ tượng trưng (symbolic size)
* Ngoài ra, KLEE cũng không thể khám phá các vũng mã chết và vùng mã không thể với tới được. (unreachable code and dead code)

### 3.2 Cascade

#### 3.2.1 Tổng quan

Cascade là một công cụ phân tích chương trình tĩnh, nhận một chương trình và một tập tin điều khiển là đầu vào. Tập tin điều khiển quy định cụ thể một hay hay nhiều assertions để kiểm tra cùng với các giới hạn trong hoạt động của chương trình. Công cụ sinh ra các điều kiện kiểm chứng cho các assertion cụ thể và kiểm tra chúng bằng cách sử dụng một SMT solver và đồng thời sinh ra một minh chứng hoặc đưa ra một ví dụ cụ thể để chỉ ra rằng assertion đó có thể sai.

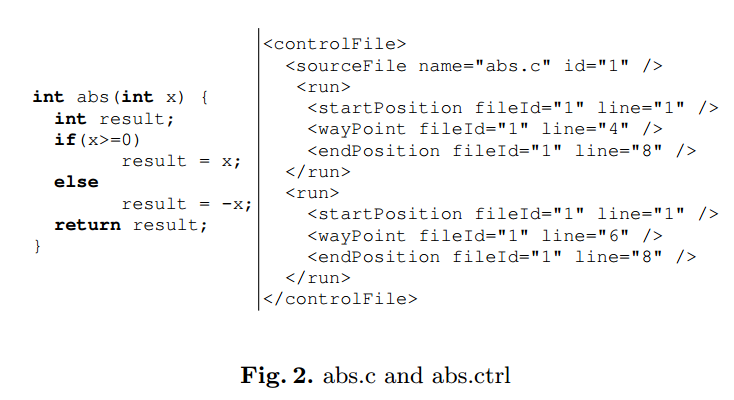
#### 3.2.2 Thiết kế hệ thống của Cascade



* Tập tin điều khiển (Control file)

Tập tin điều khiển dùng để hướng dẫn thực thi tượng trưng, sử dụng ngôn ngữ XML và hỗ trợ các cấu trúc sau.

*Cấu trúc cơ bản.* Tất cả các tập tin điều khiển đều bắt đầu với một phần tên là *sourceFile*  mà đưa ra đường dẫn tới các file nguồn. Mỗi lần chạy sẽ bắt đầu với một dòng lệnh bắt đầu (startPosition) và kết với một dòng lệnh kết thúc (endPosition). Nếu mã nguồn chứa các nhánh, Cascade sẽ xem xét cả tất cả các nhánh. Nếu người dùng muốn thực thi một nhánh cụ thể, thì có thể yêu cầu một hay nhiều lệnh (wayPoint) trong khi thực thi. Một ví dụ về tập tin điều khiển



*Lời gọi hàm.* Cascade hỗ trợ các lời gọi hàm qua nội tuyến. Nếu người dùng muốn thực thi một đường riêng biệt ở trong các hàm thì có thể sử dụng wayPoint tương tự như ở cấu trúc cơ bản.

*Vòng lặp.*  Theo mặc định, vòng lặp sẽ bị loại bỏ bằng cách tháo gỡ giới hạn vòng lặp. Một số mặc định của việc tháo gỡ vòng lặp (cho vòng lặp lặp lại một số lần nhất định) sẽ được quy định cụ thể trong dòng lệnh thực thi.

#### 3.2.3 Mô hình bộ nhớ

Một mục tiêu chính của Cascade là để hỗ trợ việc phân tích hệ thống phần mềm như các thiết bị điều khiển và xử lý mã của các hệ thống. Những chương trình này thường sử dụng rất nhiều thao tác con trỏ and đòi hỏi một mô hình bộ nhớ khá chính xác. Một mục đích khác của Cascade là mở rộng các chương trình lớn mà không phải sử dụng các con trỏ chuyên sâu. Để đạt được các mục tiêu này, Cascade cung cấp 3 mô hình bộ nhớ khác nhau với mục đích sử dụng khác nhau về độ chính xác và khả năng mở rộng:

* Flat Model

Trong đó tất cả bộ nhớ được mô hình như một mảng duy nhất

* Burstall model

Sử dụng một mảng cho mọi trường cấu trúc khác nhau

* Partition model

Chia bộ nhớ thành các phân vùng riêng biệt, sử dụng một con trỏ phân tích để đảm bảo rằng các biến ẩn có thể kết thúc trong cùng một phân vùng

### 3.3 Một số công cụ khác

# **Công cụ kiếm chứng chương trình**

## **Trừu tượng hóa chương trình dựa trên thực thi tượng trưng**

Phần lớn các công cụ kiểm chứng hiện nay thường chỉ kiểm chứng cho từng đường thi hành cụ thể của chương trình bằng cách khám phá từng đường thi hành. Công cụ được trình bày trong bản báo cáo sẽ đưa ra công thức tổng quát hóa cho cả một phương thức, từ đó có thể kiểm tra một số tính chất của phương thức cũng như là của chương trình.

Đa số các công cụ khác đều chỉ trừu tượng hóa cho một đường thi hành cụ thể nào đó trong chương trình. VTSE hướng đến việc tổng quát hóa cho toàn bộ đường thi hành vào trong một công thức cuối cùng. Làm như vậy VTSE sẽ có nhiều ứng dụng và có thể thực hiện được nhiều công việc hơn ví dụ như khám phá đường thi hành không bao giờ được thực thi hay là kiểm tra xem 2 hay nhiều đường thi hành khác nhau có cùng một công thức hay không.

### **1.1 Cây cú pháp trừu tượng (AST)**

#### 1.1.1 Giới thiệu về cây cú pháp trừu tượng

Trong việc phân tích chương trình, cây cú pháp trừu tượng (AST, Abstract Syntax Tree) đóng một vai trò rất quan trọng. AST là một cây có giới hạn, có nhãn và định hướng nhằm mục đích biểu diễn cấu trúc cú pháp trừu tượng mã nguồn của chương trình.

Ở đây, trừu tượng hóa cú pháp không phải là biễu diễn mọi chi tiết có mặt ở trong cú pháp thật của chương trình, mà nó sẽ định nghĩa các cấu trúc, các cú pháp khởi tạo.

#### 1.1.2 Cấu trúc của cây cú pháp trừu tượng

AST là một cây thể hiện các mối quan hệ của dữ liệu, nó gồm một gốc, các đỉnh và các lá. Gốc của AST thường chứa các thông tin biểu thị khởi tạo về file mã nguồn. Mỗi một nút gốc của cây được gán nhãn bằng các toán tử và các nút của cây là các toán hạng.

Mỗi một đoạn mã lệnh có thể được biểu diễn bằng một hoặc nhiều nhánh và được gắn vào trong AST, ví dụ như khối if-else sẽ được biểu diễn bằng 3 nhánh.

#### 1.1.3 Vai trò của cây cú pháp trừu tượng

AST là các cấu trúc dữ liệu được sử dụng rộng rãi trong các trình biên dịch, bởi vì khả năng biểu diễn cấu trúc mã nguồn của nó.

Một một AST thường là kết quả của quá trình phân tích và xử lý cú pháp của trình biên dịch. Nó thường được sử dụng như là một biểu diễn ngay tức thời của chương trình mà trình biên dịch yêu cầu và có tác động lớn đến kết quả cuối cùng của trình biên dịch.

AST thể hiện bản chất vốn có của các ngôn ngữ lập trình và tài liệu liên quan. Ở công cụ kiểm chứng chương trình được trình bày trong bài báo cáo này, việc trừu tượng hóa chương trình bắt đầu từ việc xây dựng cây cấu trúc cú pháp trừu tượng. Việc này nhằm mục đích xác định các thành phần của mã ngồn, cấu trúc của nó cũng như sự liên kết giữa các thành phần trong mã nguồn.

Trong việc trừu tượng hóa chương trình, cây AST dùng để xây dựng nên đồ thị dòng điều khiển (Control Flơ Graph), là bước đầu tiên để phân tích chương trình cũng như dựa vào đó để lấy được các thành phần cấu tạo nên chương trình. Việc xây dựng cây AST trong công cụ này sẽ được trình bày cụ thể trong mục 1.1 (phần IV. Thực nghiệm)

#### 1.1.4 Ví dụ về cây cú pháp trừu tượng



**Mã nguồn:** Giải thuật Euclid

### **1.2 Xây dựng đồ thị luồng điều khiển**

#### 1.2.1 Giới thiệu về đồ thị luồng điều khiển

Đồ thị luồng điều khiển (Control Flow Graph, CFG) là một đồ thị có hướng mô tả cấu trúc logic của chương trình một cách trực quan và đơn giản hơn, gồm có các đỉnh tương ứng với các câu lệnh/nhóm câu lệnh và các cạnh là các dòng điều khiển giữa các câu lệnh/nhóm câu lệnh. Đỉnh đầu tiên của CFG là trạng thái đầu tiên của hàm, đỉnh cuối cùng là trạng thái kết thúc của hàm. Đỉnh i nối đến đỉnh j thì câu lệnh tương ứng đỉnh j có thể được thực thi sau khi thực hiện câu lệnh tương ứng ở đỉnh j.

#### 1.2.2 Cấu trúc, các thành phần của CFG

Trong một CFG điển hình, mỗi đỉnh sẽ biểu thị một khối lệnh cơ bản, mỗi một đường nối các đỉnh trong đồ thị sẽ biểu diễn sự chuyển tiếp giữa các khối lệnh. Trong đa phần các đồ thị, có hai khối lệnh điển hình, đó là khối lệnh thực thể và khối kết thúc.

CFG là một đồ thị có hướng, các chu trình thể thiện vòng lặp và mỗi một đường đi trong từ đỉnh đầu đến đỉnh cuối của nó thể hiện sự mô phỏng hóa một đường thi hành cụ thể có trong chương trình.

#### 1.2.3 Chức năng của CFG

CFG là một thành phần rất quan trọng và cần thiết. Nó được sử dụng rộng rãi trong nhiều công cụ về kiểm chứng, kiểm thử, phân tích chương trình và các trình biên dịch.

CFG cho ta thấy được cái nhìn tổng quan về các đường thi hành có trong một chương trình. Từ đó ta có thể nắm bắt được cách mà chương trình sẽ thực thi và cho kết quả.

Trong công cụ này, ta cũng sử dụng CFG để có thể nắm bắt được tất cả các đường thi hành trong chương trình, để sau đó tiến hành gỡ các chu trình lặp của đồ thị và đưa ra công thức tổng quát hóa của nó.

Ở trong công cụ, CFG được xây dựng từ AST (cụ thể sẽ được trình bày trong phần 2.2 phần IV.Thực nghiệm)

#### 1.2.3.4 Ví dụ về CFG

#### https://upload.wikimedia.org/wikipedia/commons/thumb/3/30/Some_types_of_control_flow_graphs.svg/270px-Some_types_of_control_flow_graphs.svg.png

Các vị dụ về CFG

### **1.3 Loại bỏ chu trình trong CFG**

Một vấn đề nảy sinh khi mà đồ thị luồng điều khiển chứa các chu trình đơn do chương trình có các đoạn mã sinh vòng lặp (for, while và do-while). Các vòng lặp thường lặp với một số lần không cố định trong mỗi lần thực thi. Điều này gây khó khăn trong việc xác định quá trình thực thi của chương trình sẽ được thực hiện như thế nào.

Bài toán xử lý và phân tích vòng lặp là một bài toán khó, phức tạp trong các bài toán phân tích chương trình và tính đến hiện nay chưa có một giải pháp nào triệt để cho vấn đề này.

Trong công cụ VTSE chúng tôi đề xuất ra một kĩ thuật để có thể gỡ bỏ được các chu trình có trong CFG. Kĩ thuật này được gọi là unfold CFG, và được tiến hành ngay sau khi tạo được CFG từ AST. Mục tiêu chính của kĩ thuật này là tiến hành nhân bản một phần của CFG được tạo bởi khối lệnh lặp lên dựa vào số lần lặp thực tế trong chương trình hoặc số lần lặp nào đó do chính người dùng yêu cầu. Nếu không xác định được số lần lặp thì sẽ tiến hành một số lần lặp tương đối dựa vào tính chất của chính khối lệnh này.

Chi tiết về kĩ thuật unfold CFG sẽ được trình bày trong mục 1.3 phần IV. Thực nghiệm

Ví dụ về unfold CFG

### **1.4** **Đánh chỉ số (indexing) cho các tham biến và xây dựng biểu thức metaSMT**

#### 1.4.1 Đánh chỉ số cho các biến

Sau khi xây dựng được đồ thị luồng điều khiển cho phương thức, ta sẽ xây dựng công thức tổng quát “sơ khai” cho phương thức, được gọi là biểu thức metaSMT. Ở bước này ta sẽ đánh chỉ số (indexing) cho các tham biến có mặt trong phương thức.

Mỗi lần xuất hiện của biến ở vế trái của biểu thức gán sẽ được đánh dấu bằng một chỉ số tăng dần của biến. Chúng ta phải làm điều này bởi vì trong các SMT Solver, mỗi một biến sau khi được gián trị thì sẽ không thể thay đổi được giá trị của nó. Mà trong chương trình thật, một biến thường không có một giá trị cụ thể từ đầu cho đến cuối chương trình. Chính vì vậy mà tiến hành đánh chỉ số cho các biến của chương trình là một việc làm cần thiết.

Riêng đối với khối lệnh điều kiện if-else hoặc switch-case, một biến có thể xuất hiện ở nhánh này nhưng lại có thể không xuất hiện ở nhánh kia. Để giải quyết vấn đề này, ta phải tiến hành đồng bộ chỉ số của các biến xuất hiện trong khối lệnh if-else theo biến có chỉ số cao nhất. Chi tiết sẽ được trình bày trong mục 1.4 phần IV. Thực nghiệm khi tiến hành cài đặt công cụ.

#### 1.4.2 Công thức metaSMT

Kết thúc việc đánh chỉ số cho toàn bộ các biến có mặt trong phương thức, ta sẽ đưa ra dạng công thức sơ khai của công thức tổng quát hóa, đó là công thức metaSMT.

Đưa ra công thức metaSMT là bước đệm trước khi đưa ra công thức tổng quát hóa cuối cùng của phương thức. Công thức metaSMT cũng gần giống như công thức tổng quát hóa cuối cùng là bao gồm công thức của tất cả các nhánh thi hành có thể có của chương trình, chỉ khác là ở dạng metaSMT biểu thức sẽ gần giống với biểu thức ở câu lệnh thật và dễ quan sát cũng như dễ hiểu hơn so với công thức tổng quát hóa cuối cùng.

Nhiệm vụ chính của metaSMT là giúp chúng ta kiểm soát lỗi và nắm bắt được dạng của công thức cuối cùng, giúp cho việc trừu tượng hóa được minh bạch và dễ kiểm soát hơn.

Ví dụ về đánh chỉ số và công thức metaSMT

### **1.5 Công thức tổng quát cho phương thức**

Sau khi đã được ra được metaSMT của phương thức, ta sẽ đưa ra được công thức tổng quát của phương thức. Công thức tổng quát được đưa ra dựa trên metaSMT của phương thức và được biểu diễn dưới dạng tiền tố của các biểu thức logic vị từ cấp 1 (First-order logic).

Đây chính là công thức mong muốn của chương trình, là đưa ra được công thức tổng quát cuối cùng của phương thức. Công thức này bao gồm tất cả các đường thi hành của chương trình

Công thức tổng quát hóa cuối cùng được biểu diễn bằng các biểu thức logic cấp 1 dưới dạng tiền tố và sử dụng các chuẩn của SMT Solver.

Ví dụ về công thức cuối cùng

### **1.6 Kết hợp điều kiện của người dùng và đưa vào bộ giải**

Sau khi đưa ra được công thức tổng quát cho phương thức, ta kết hợp với điều kiện mà người dùng nhập vào để đưa vào bộ giải SMT Solver và đưa ra kết quả cuối cùng cho người dùng. Ở đây, ví dụ người dùng muốn kiểm tra xem hàm f có luôn nhận về giá trị lớn hơn 0 hay không, người dùng nhập vào điều kiện: f > 0

Từ đó ta có ràng buộc cuối cùng để đưa vào bộ giải đó là: constraints = F ^ (f > 0)

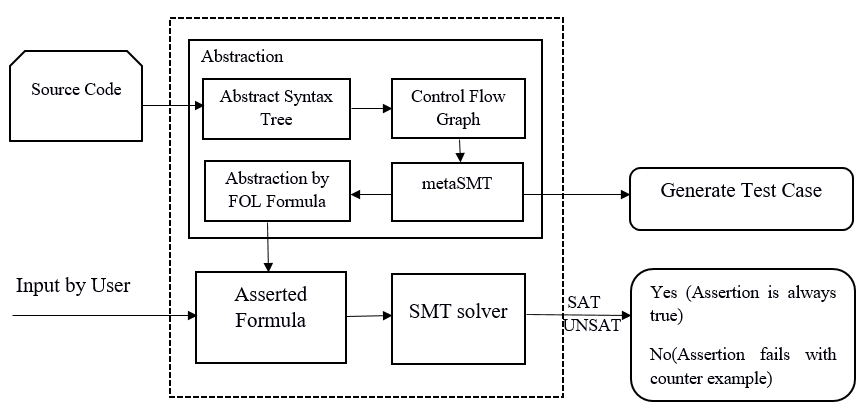
Nếu kết quả trả về là SAT thì tức là có ít nhất 1 trường hợp làm cho f <= 0, công cụ sẽ đưa ra một ví dụ để phản ví dụ cho trường hợp này

Nếu kết quả trả về là UNSAT thì tức là f > 0 với mọi giá trị đầu vào a và b

Sau đây, bài báo cáo sẽ đi vào chi tiết từng phần để tổng quát hóa công thức của chương trình.

## **Framework**

Để làm được điều này, chương trình sẽ được thực hiện theo một mô hình để tổng quát hóa từ mã nguồn thành công thức tổng quát hóa cuối cùng như sau:



## **Case Study**

Để minh họa cho việc thực hiện chương trình, chúng tôi xin đưa ra 1 ví dụ cụ thể. Giả sử ta có một phương thức được viết bởi ngôn ngữ Java như sau:

public static int f(int a, int b){

int result = 0;

if(a < b ){

a = b + 1;

result = a + b;

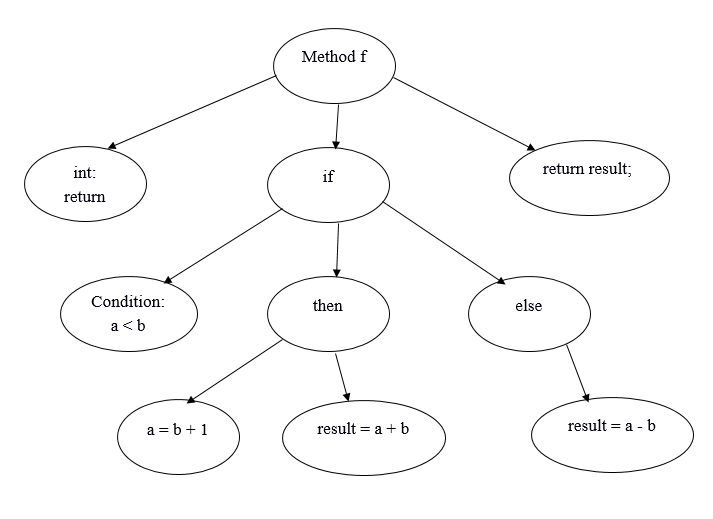
}else

result = a – b;

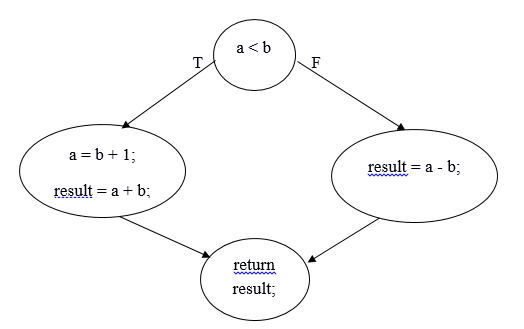
return result;

}

Bước 1: Xây dựng cú pháp cây trừu tượng. Từ đoạn mã nguồn của phương thức trên ta xây dựng được cây cú pháp trừu tượng như sau:



* Bước 2: Xây dựng đồ thị luồng điều khiển từ cây cú pháp trừu tượng đã xây dựng ở bước 1. Đồ thị luồng điều khiển của phương thức trên có dạng như sau:



* Bước 3: Diễn giải CFG nhằm để diễn giải vòng lặp nếu có của phương thức. Ở case study này phương thức sử dụng không có chứa vòng lặp. Vì vậy mà không cần phải tiến hành bước diễn giải CFG
* Bước 4: Đánh chỉ số (indexing) cho các tham biến và xây dựng biểu thức metaSMT

Sau khi đã diễn giải CFG, ta tiến hành đánh chỉ số cho các tham biến và đưa ra biểu thức metaSMT

/\* metaSMT function f \*/

/\* begin \*/

int result\_0 = 0;

if(a\_0 < b\_0 ){

a\_1 = b\_0 + 1;

result\_1 = a\_1 + b\_0;

}

else{

result\_1 = a\_0 – b\_0;

a\_1 = a\_0;

}

return result\_1;

/\* end \*/

* Bước 5: Xây dựng công thức tổng quát cho phương thức:

Ta đưa ra công thức tổng quát cho phương thức từ biểu thức metaSMT:

F = (result\_0 = 0)

^ ((a\_0 > b\_0) => (a\_1 = b\_0 + 1) ^ (result\_1 = a\_0 + b\_0))

^ (-(a\_0 > b\_0) => (result\_1 = a\_0 – b\_0))

^ (return result\_1)

* Bước 6: Kết hợp điều kiện người dùng và đưa vào bộ giải

Sau khi có được công thức tổng quát cho phương thức, ta đưa công thức vào bộ giải SMT Solver cùng với các bộ điều kiện do người dùng nhập vào. Từ đó dựa vào kết quả trả về, ta đưa ra kết luận về bộ điều kiện đó.

Ví dụ: Giả sử người dùng nhập vào bộ điều kiện: f > 0. Tức là người dùng muốn kiểm tra xem hàm f có luôn nhận giá trị trả về là một giá trị luôn dương hay không ? Khi đó, bộ ràng buộc cuối cùng được đưa vào bộ giải là: F ^ f.

Kết quả, khi đưa vào bộ giải SMT Solver, ta nhận được kết quả là UNSAT, cùng với đó là giá trị f = -4 và một cặp giá trị (a, b) là (-3, -2). Tức là mệnh đề f>0 bị sai với bộ giá trị cụ thể là (-3, -2), hay nói cách khác giá trị trả về của hàm f có thể mang giá trị không dương.

# Thực nghiệm

## Công cụ

### 1.1 Tạo cây trường tượng (Abstract Syntax Tree) từ mã nguồn

AST được tạo từ 1 công cụ mã nguồn mở là Spoon

### 1.2 Tạo đồ thị luồng điều khiển (Control Flow Graph – CFG) từ cây cú pháp trừu tượng

CFG là gì. Cấu trúc của 1 cfg, nó có tác dụng gì, tại sao lại cần cfg trong công cụ, nó được xây dựng từ đâu (từ cây AST), có nhiệm vụ gì cho các phần tiếp theo (metaSMT)

Phần này sẽ giới thiệu chi tiết các thành phần của CFG, các loại Node: Begin, End, Node condition và node chứa câu lệnh bình thường. Biểu diễn CFG cho từng đoạn riêng biệt: if-else, switch, for. Riêng for thì nêu ra ví dụ.

### Unfold cfg

Nói là có cho bao nhiêu vòng lặp mặc định, và cho phép người dùng nhập vào số vòng lặp

Unfold cfg là gì, nó là một quá trình dễ diển giải cfg thành một đồ thị tuần tự, k có chu trình. Kĩ thuật unfold cho vòng lặp, nêu ví dụ

### Đánh chỉ số (indexing) cho các biến và tham số trong phương thức (metaSMT)

Tổ chức đồng bộ chỉ số, meta

### 1.5 Đưa ra công thức tổng quát

### 1.6 Đưa vào bộ giải và nhận kết quả

## Một số ứng dụng thực nghiệm

Công cụ kiểm chứng tính chất của chương trình có khả năng đem lại cho chúng ta rất nhiều các ứng dụng khác nhau trong việc kiểm tra tính chất, sự thỏa mãn một tính chất nào đó của các phương thức trong chương trình.

Kiểm tra được các tính chất của chương trình đem lại rất nhiều lợi ích trong sự kiểm soát lỗi và gỡ lỗi của chương trình. Điều này là rất cần thiết đối với các nhà phát triển cũng như một người lập trình viên.

Ứng dụng của chương trình có phạm vi rất lớn trong các công việc xử lý nặng về tính toán, đặc biệt là các chương trình có nhiều câu lệnh điều khiển rẽ nhánh và biến đổi phức tạp.

Ta sẽ kết hợp điều kiện của người dùng muốn kiểm tra với công thức tổng quát của chươn trình để tạo thành một chuỗi các ràng buộc cuối cùng và đưa vào bộ giải để đưa ra kết luận.

### 2.1. Kiểm tra giá trị trả về có thỏa mãn điều kiện

Ứng dụng đầu tiên của công cụ kiểm chứng tính chất của chương trình mà chúng tôi muốn đề cập đến đó là kiểm tra xem giá trị trả về của một phương thức hay giá trị của một tham biến nào đó có trong chương trình có thõa mãn điều kiện nào đó hay không.

Trong rất nhiều bài toán, việc kiểm tra giá trị trả về của một phương thức hay một hàm có thỏa mãn một điều kiện nào đó là một việc rất cần thiết và quan trọng. Tuy nhiên, để làm được điều này lại là một việc khó khăn. Công cụ kiểm chứng tính chất của chương trình – VTSE có khả năng trừu tượng hóa công thức một cách tổng quát, biến đổi từ các dòng lênh của của chương trình thành một công thức tổng quát duy nhất. Do đó ta có thể dễ dàng kiểm tra được giá trị của phương thức có thỏa mã một đẳng thức hay một bất đẳng thức nào đó hay không.

Ở trường hợp này, người dùng sẽ nhập vào một đẳng thức hay bất đẳng thức của giá trị trả về. Ta kết hợp công thức tổng quát của phương thức với điều kiện do người dùng nhập này để đưa vào bộ giải.

Ví dụ: người dùng muốn kiểm tra giá trị trả về của một phương thức hay một hàm có luôn dương hay không, thì biểu thức cần nhập vào là: f > 0, ở đây f là biến tượng trưng cho giá trị trả về của phương thức.

Giả sử công thức tổng quát hóa cuối cùng là F, ta có ràng buộc cuối cùng để đưa vào bộ giải đó là: F ^ f. Sau khi đưa ràng buộc vào bộ giải, ta có thể nhận về một trong hai giá trị là YES/NO. Nếu kết quả là YES thì phương thức cần kiểm tra luôn nhận một giá trị trả về lớn hơn không, còn nếu ngược lại là NO thì sẽ đưa ra một phản ví dụ cho thấy rằng f sẽ nhận giá trị nhỏ hơn hoặc bằng không với một bộ giá trị nào đó.

### 2.2. Xác định khoảng giá trị trả về của phương thức

Ứng dụng thứ 2 của công cụ mà chúng tôi muốn đề cập đến đó là xác định được khoảng giá trị trả về của phương thức hay kiểm tra phương thức đó luôn có giá trị trả về nằm trong một khoảng cho trước nào đó hay không?

### 2.3. Kiểm tra tính chất của các bộ giả định