# General Info / Home Page

## Short Bio

She is a PhD Student at Imperial College London under the supervision of Prof. Philippa Gardner and a member of the Program Specification and Verification Group. She is currently doing research on formal and automated verification about JavaScript programs. Her areas of interest include Web Development, Software Product Lines, Software Reuse as well as Functional and Object-oriented Programming Languages.

She received a B.Sc. and a M.Sc. in Computer science from Federal University of Pernambuco in Brazil. She interned as a software developer at SUATI, and later became a web developer at KOY, a technological company that aims to integrate software solutions into the legal system. During her internship and career as a web developer, she became aware of the lack of automated verification of JavaScript programs. Dedicated to helping other web developers in the future, she decided to join the team at Imperial College London to research automated testing and verification of JavaScript, which will potentially increase the productivity and efficiency of many others working in the field of web development.

She also enjoys travelling, running and doing Pilates in her free time.

## Publications

Her publications are shown as follows:

* "[Partially safe evolution of software product lines](http://delivery.acm.org/10.1145/2940000/2934482/p124-sampaio.pdf?ip=129.31.158.39&id=2934482&acc=ACTIVE%20SERVICE&key=BF07A2EE685417C5%2EF5014A9D3D5CC2D9%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1549202041_07d35732f8c5a93db8a35d4a1ef7e48e)." In Proceedings of the 20th International Systems and Software Product Line Conference, pp. 124-133. ACM, 2016.
* “[JaVerT 2.0: Compositional Symbolic Execution for JavaScript.](http://delivery.acm.org/10.1145/3300000/3290379/popl19main-p189-p.pdf?ip=129.31.158.39&id=3290379&acc=OA&key=BF07A2EE685417C5%2EF5014A9D3D5CC2D9%2E4D4702B0C3E38B35%2E6D218144511F3437&__acm__=1549202435_d9b318b136e17b3715f3fa5a9e3c6e56)” PACMPL, vol. 3 (POPL).
* "Compositional Symbolic Analysis for JavaScript: Unifying Symbolic Execution and Separation Logic." POPL, 2019.

## Teaching

She is currently helping with two courses at Imperial College London:

**Models of Computation (2nd year course)**

This course covers key concepts in theoretical computer science, such as operational semantics, register machines and Turing machines.

**Software Engineering Design (2nd year course)**

This course is about software design and covers, among other topics, design principles, design patterns, architectural styles and concurrency.

# Automatic JavaScript Program Verification

## Introduction

Since this research centers around JavaScript, the programming language itself aught to be introduced first before the whole research can be discussed.

### JavaScript

JavaScript is one of the most popular programming languages used to form the World Wide Web as we know it today. It is a “client-side” programming language, which means that the code is read, interpreted and executed in the client’s web browser. The role which JavaScript plays is essential as it enables the user to interact with web pages. Simply put, without JavaScript, most of the websites today would be stagnant and boring, and we would not be able to see the features that make our lives more convenient, such as interactive maps and animated graphics. Its popularity can be demonstrated by the fact that nowadays, around 94.9% of the websites use JavaScript, and it is also the most active language in both GitHub and StackOverflow.

### Objective

As now we have gained a general idea of what JavaScript does, it is time to inspect the overall objective of this research.

When a piece of code is written, it is highly likely that there will be bugs, which may have unwanted and significant impacts on the functionality as well as the correctness of the programs. To help programmers eliminate as many bugs as possible, various tools have been created. For instance, a static code analysis tool, called Infer, was developed by Facebook in 2013, and it is used to find bugs for languages such as C, C++, Java and Objective C. With a tool like Infer, hundreds of bugs are identified every month before the code is finalized.[[1]](#footnote-1) This saves the programmers a large amount of time finding and fixing bugs, which is a tremendous increase in productivity.

JavaScript, despite its popularity and ubiquity, has not had a verification tool designed for it. This is mainly caused by its flexible features such as dynamic typing, which adds a lot of complexity to the construction of the verification program. In order to develop a similar verification tool like Infer for JavaScript programmers, a program called **JaVerT** (JavaScript Verification Toolchain) was conceived by Ms. Sampaio and her team. The goal of this research project is to investigate the effectiveness of JaVerT as well as the components and principles on which it is based.

## Main Components

There are numerous which contributed to the solid foundation of JaVerT. The few main ones are separation logic, symbolic execution, bi-abduction and JSIL. For those that are interested in the technical details, a short description for each component is listed as follows:

### Separation Logic

Different types of logic are created over the past in order for us to reason about different things. For example, there are propositional logic, which is used to describe simple propositions; first-order logic, which extends propositional logic by adding variables and quantifiers[[2]](#footnote-2). Separation logic, expectedly, was designed to deal with things which are not included in other types of logic. To be more specific, it enables us to reason about[[3]](#footnote-3):

* **Shared and mutable data structures**: structures where updatable fields can be referenced from more than one point.
* **Local reasoning**: specifications and proofs for a program component refer only to the memory that component accesses.
* Recently, it was extended to incorporate **concurrency** and **information hiding**.

With the use of separation logic, JaVerT is able to handle the dynamic and flexible features in JavaScript.

### Symbolic Execution

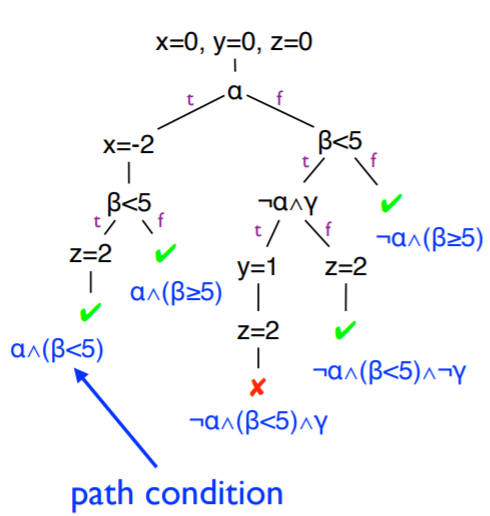
In order to tell if a program is written correctly, different test cases are usually created. For instance, to see if a function outputs an integer smaller than 5 no matter what the input is, we can write the following:

.

However, as one notices here, it only tests for one possible situation, which is when the input of the function is 2. It does not cover all the cases described in the criterion, namely the output should be an integer less than 5. To generalize the testing, symbolic execution is used. For example, in this case, instead of tediously writing out test cases for all the inputs, we can just say:

In this case, we replace the specific number in the input by unknown variables. Moreover, when an “if” statement (a type of statement that checks if a condition is met) is seen in the program, different paths can be generated for each condition in the statement. The overall result of the symbolic execution can be represented by a flow graph, an example of which is shown as follows:

**Figure 1**: Symbolic execution flow graph[[4]](#footnote-4)



Each path represents many actual test cases, and therefore, we are able to cover more situations so as to check the correctness of the program.

The concept of symbolic execution is applied to many domains, such as verification, testing, debugging and optimization, and parts of JaVerT’s functionalities are realized with the help of this power concept.

### Bi-abduction

Abductive inference is a concept from Philosophy and means finding the most likely explanation from an observation. Bi-abduction, simply put, allows the pre and post conditions to be generated from the bare source code[[5]](#footnote-5). Here, pre-condition refers to the constraints or statements that must be satisfied before a program is called, and similarly, post-condition describes the statements that will be true after a program is executed. In short, bi-abduction provides JaVerT with the capability to see what the input and output of a piece of code should roughly look like.

### JSIL

## Examples

## Potential Impacts

## Related Work

### Other Tools

# Future Goals

# References

1. https://www.informationweek.com/mobile/mobile-applications/facebook-infer-bug-detection-tool-is-now-open-source/d/d-id/1320847 [↑](#footnote-ref-1)
2. http://www-verimag.imag.fr/~serban/talks/2014.12.10-intro-sl.pdf [↑](#footnote-ref-2)
3. https://www.cs.cmu.edu/~jcr/copenhagen08.pdf [↑](#footnote-ref-3)
4. https://www.cs.umd.edu/~mwh/se-tutorial/symbolic-exec.pdf [↑](#footnote-ref-4)
5. https://fbinfer.com/docs/separation-logic-and-bi-abduction.html [↑](#footnote-ref-5)