# Improving Conditional Inference in Daikon

Aidan Shackleton (ashackl1@swarthmore.edu) and Sophie Libkind (slibkin1@swarthmore.edu)

**Abstract**

Finding conditional invariants is hard. We examine Daikon, an existing system that dynamically detects conditional invariance, evaluating its output on a customized set of programs. We then propose and implement a series of modifications to Daikon's inference generator and variable grammar with the goal of improving its conditional invariance detection without compromising performance.

## 1 Motivation

Conditional invariants present a challenge for software engineers. In nontrivial programs, including iterative, recursive, and parallelized algorithms, they can be critical to establishing correctness, robustness, and general behavior, yet often their functions within a program are loosely documented and poorly understood. As part of its dynamic invariant detection process, Daikon [3] implements a system for detecting conditional invariants. This system has not been subjected to thorough evaluation, however, and could likely stand to be improved. Given the complicated nature of conditional invariants and their importance to programmers, perfecting Daikon's conditional property inference would be a step towards better software development and maintenance.

## 2 Background

Given a program, an *invariant* is a property held at a particular point or set of points during the program's execution. For instance, "$x \neq 0$," "`sizeof(y)>sizeof(x)`", and "$ls$ is sorted" are all invariants. A *conditional invariant* is an invariant that depends on some other value or property: "if $x = 0$, the function returns an empty list" and "after $n$ loop iterations, the first $n$ elements of the list are sorted" are both conditional invariants.

Daikon uses dynamic instrumentation to track variable values, feeding the result into an inference generator that creates a variable grammar and applies a machine learning process to derive a set of likely invariants. Daikon allows the user to specify a customizable class of invariants to track, in addition to providing a set of default invariant options.

In order to determine conditional invariance, Daikon relies upon a set of splitters, which are predicates to conditional statements. Each splitter precipitates two conditional program points, creating one data trace for program executions that satisfy the predicate and one for program executions that do not. Based on the invariance determined for each data trace, Daikon can infer conditional invariance of the form "$p \Rightarrow q$" where $p$ is the predicate that split the data trace and $q$ is an invariant that holds for the data trace corresponding to $p$ and not on the unsplit data trace. Furthermore, if Daikon detects the same invariance for multiple splits (for example, $p \Rightarrow q$ and $p' \Rightarrow q$), it can infer biconditional statements ($p \iff p'$).

Daikon's ability to infer conditional invariance relies on two key factors. First, Daikon's grammar must be expressive enough to articulate desirable splitters. Second, Daikon must either rely on user inputted

splitters or must have a technique for determining splitters. Daikon currently supports three techniques for automatically choosing splitters:

**Indeterminant splitting:** In this method every conditional statement is treated as a splitter. This has the advantage of considering every useful predicate allowable by Daikon's grammar at multiple program points. However, indeterminant splitting creates significant slowdown as it attempts to use conditionals that are not appropriate at certain program points.

**Clustering:** Daikon creates clusters of similar data points using the k-means algorithm, hierarchical clustering, or the x-means algorithm. Daikon determines invariants on each cluster. If an invariant occurs in one cluster but not in others then it is considered a conditional invariant where the predicate is membership in the cluster.

**Random selection:** Daikon randomly selects several small subsets of data from original trace. Invariants discovered in one of the subsets but not in the whole file are considered conditional invariants where the predicate is membership in the subset.

In [1], the authors found that the clustering and random techniques performed similarly in the contexts of program verification and error detection. There is certainly still room for improvement, however. It was noted in the same paper that "In some cases, such as error detection in *print_tokens2*, no policy worked well, and future research is required." Michael Ernst, one of the authors of Daikon and many associated papers [1, 2, 3], suggested in correspondence with us that continuing work on Daikon's conditional inference mechanism is important.

# 3 Our Idea

We plan to submit Daikon to a rigorous battery of tests, evaluating its conditional inference, locating errors and time sinks, and determining ways to improve the current system. We will focus on two aspects of conditional invariance detection: grammar expressiveness for predicates and techniques for splitter selection. The first phase will involve evaluating both aspects by testing programs with known conditional invariance.

The results of our evaluation will determine the trajectory of the remainder of our project. The our goal will be to improve upon Daikon's conditional invariance detection system based on the flaws exposed by our evaluation. Likely improvements may include:

- Proposing an alternative and more tailored grammar for defining conditional predicates.
- Implementing a learning algorithm to suggest or choose the most appropriate technique for splitter selection, given a specific program.
- Implementing a new technique for splitter selection, either by aggregating previous methods or by analyzing and solving gaps in Daikon's current system.

# 4 Milestone Goals

For Milestone 1, we will investigate the expressiveness of Daikon's grammar with respect to predicates of conditional invariance. Our goal is to identify a class of unarticulatable predicates and investigate the limitations of using an identical grammar to identify predicates and invariants.

For Milestone 2, we will carefully analyze Daikon's three techniques for choosing splitters in the context of determining conditional invariants. We will evaluate each technique based on how well it performs under different types programs with known conditional invariance.

For Milestone 3, we will propose solutions to flaws exposed by our evaluations. This milestone may include proposing a new grammar for defining predicates or implementing a new technique for splitter selection.

# References


[1] Nii Dodoo, Lee Lin, and Michael D Ernst. Selecting, refining, and evaluating predicates for program analysis. 2003.

[2] Michael D Ernst, Adam Czeisler, William G Griswold, and David Notkin. Quickly detecting relevant program invariants. In *Proceedings of the 22nd international conference on Software engineering*, pages 449–458. ACM, 2000.

[3] Michael D Ernst, Jeff H Perkins, Philip J Guo, Stephen McCamant, Carlos Pacheco, Matthew S Tschantz, and Chen Xiao. The daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1):35–45, 2007.