# Tamarin: Concolic Program Disequivalence for MIPS

Abel Nieto

University of Waterloo
anietoro@uwaterloo.ca

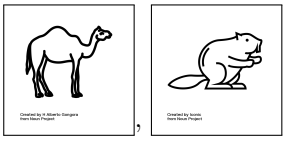
**Abstract.** TODO

## 1 Introduction

We are staring at two opaque black boxes laying at our feet. Each box has a narrow slot through which we can place items in the box, but we cannot quite see what is inside. They look approximately like this:

We know each box contains an animal, but we do not know which specific animal is in each one. We would like to find out if both boxes contain the same species of animal. Our solution is simple: we take two carrots, and drop one in each box through the slots.

After a while, a chewing sound emerges from the boxes. We peer into them and, indeed, it looks like the carrots were successfully eaten. Triumphantly, we declare that the boxes contain the same species of animal. The truth is altogether different:



The boxes are assembly programs. The animals are the functions those programs compute. The carrot is unit testing. The task was to determine whether the programs were equivalent. And we failed at it. In this paper, we show a technique that is better than the carrot.

Program equivalence. The program is the specification. The complications of assembly language.

## 2 Program Disequivalence for MIPS

Let us set up the problem a bit more formally. Consider the set $P$ of MIPS-assembly programs that satisfy two restrictions: they take as inputs only the

values of registers \$1 and \$2, and when they stop executing we define their output to be (exclusively) the value of \$3. Other side effects, such as printing values to the screen or system calls, are disallowed.

We can now define a relation $\mathsf{equiv} \subseteq P \times P$ of equivalent programs. Given $P_1, P_2 \in P$, we say that $P_1$ $\mathsf{equiv}$ $P_2$ (read "$P_1$ is equivalent to $P_2$") if $P_1$ and $P_2$ produce the same output (the same value of \$3), respectively, for all inputs (for all values of \$1 and \$2). For example, the two following programs are equivalent:

```
# P_1
add $3, $1, $2
```

```
# P_2
add $4, $1, $1
lis $5
42
sw $4, 0, $5
add $3, $1, $2
```

Notice that $P_1$ $\mathsf{equiv}$ $P_2$ even though $P_2$ modifies the contents of the memory and an additional register (\$4), because:

- Both $P_1$ and $P_2$ terminate without errors.
- The value of \$3 will be the same when they do so.

Unfortunately, even though $\mathsf{equiv}$ captures an already-simplified notion of equivalence[1], a decision procedure for it does not exist, due to Rice's theorem.

To get decidability back, we define a new class of relations $\mathsf{equiv}_S \subseteq P \times P$. We say that $P_1$ $\mathsf{equiv}_S$ $P_2$, read "$P_1$ is potentially equivalent to $P_2$ up to $S$ steps" if

## 3 Concolic Program Disequivalence

## 4 Tamarin

Overview.

### 4.1 Trace Collection

CPU instrumentation, PC concretization, error boxing, and fuel.

### 4.2 Transformations

Desugaring, simplification, trimming, and conversion to SSA.

### 4.3 Query Representation

Memory, jumps, arithmetic operators.

[1] For example, $\mathsf{equiv}$ has a very narrow notion of output that excludes side effects.

### 4.4 Concolic Execution Redux

Alternation. Compatibility. Soundness/Completeness. Efficiency.

## 5 Evaluation

## 6 Related Work

## 7 Conclusions