# STM Project Proposal

Hitarth
Bhishmaraj

MSc CS Students, CMI

October 24, 2019

# 1 Introduction

Solver-aided programming language/framework, such as Rosette [1], extend traditional programming languages with SAT/SMT-specific interface and constructs. Such a language framework makes it easier to embed/model domain-specific artifacts/systems and exploit use of SAT/SMT solver features (UNSAT, MAX-SAT, UNSAT-CORE, etc.) for performing various constraint-solving tasks, such as symbolic verification, debugging, bug localization, and synthesis. Most of the current work in this field is focussed on arithmetic and bit-vector theories. There are tools for verification of programs in ANSI C with suitable assertions to a limited extent, like BugAssist[2], but they don't focus on other solutions like synthesis. Also it's code is not open source.

**In this project** we propose to use Rosette-Racket for analysis (*verification*, *debugging*, and *bug-fixing*) of array manipulating programs. An array theory poses a challenge for symbolic analysis as it is undecidable, in general, because it requires quantifier instantiation. We simplify our problem by restricting ourselves to a decidable fragment of arrays theory [3]. We simplify the problem of bug-fixing, which is essentially a synthesis problem, by restricting the grammar of the expressions that can be used in the fixes.

# 2 Problem: Automatic Verification, Debugging and Fixing Array Programs

Consider a simple program that is expected to swap the values at $i$ and $j$ index of an array $a$ if they are not in ascending order. We have deliberately introduced a **bug** in line 5 of the program by using $j$ instead of $i$ in the array select to preserve the post-assertion

```
1: int[10] a;
2: unsigned int i, j;
3: @Pre : assume(i < 10 && j < 10)
4: if (a[i] <= a[j]) {
5:     temp = a[j]; //Bug!!
6:     a[i] = a[j];
7:     a[j] = temp; }
8: @Post : assert(a[i] > a[j])
```

Our goal is to develop a prototype tool that does the following: (1) *verify* such program assertions; if an assertion fails (2) localize *bugs* to a region (line 5) of the program, and suggest a possible fix (replace $J$ by $i$) to make the assertion true.

# 3 Approach

Srivas: Merge this part with Approach; Note that for SMT proposal you have a limit of 2 pages.

Srivas: You can include an extended version of the grammar I have shown for Problem 3 of assignment to include array selects and updates.

Srivas: You should specify the class of fixes you will be restricting yourself to.

We will covert the program in logical formula using Racket/Rosette, and then proceed to the verification. Assume that the final formula we get is $P$.

$$\begin{aligned}
&P \coloneqq (\\
&(a_i = a[i] \wedge a_j = a[j])\\
&\wedge (a[i] > a[j] \implies t = a[i]\\
&\wedge a' = a\{i \leftarrow a[j]\})\\
&\wedge a'' = a'\{j \leftarrow t\}\\
&)\\
&\implies ((a_i > a_j \implies a[i] = a_j \wedge a[j] = a_i)\\
&\wedge (\neg(a_i > a_j) \implies a[i] = a_i \wedge a[j] = a_j))
\end{aligned} \tag{1}$$

**Verification:** We expect that $\neg P$ will be $UNSAT$. If it is the case, the program is verified.

**Debugging:** If $\neg P$ is $SAT$, then there is a bug in the program. We shall get the model for this $SAT$ instance, and check for the $UNSAT$ core for $P$ under this model. This, we expect, will be done with the help of Rosette.

**Synthesis:** For the sake of simplicity, **we shall assume that the bug lies in the array access operations** in the program. For synthesis, we shall convert the program to a sketch by introducing *holes* in the array access operations. Then, with the help of Rosette, we shall try to find out the possible substitution for the holes so that the $\neg P$ becomes $UNSAT$, and hence the program becomes correct.

1. Describing a language which can be used to specify the array manipulating programs with the pre/post conditions and loop invariants

2. Develop an interpreter for this language using Rosette which will help in converting the problem of performing the following analysis of a restricted class of array manipulating programs into instances of SMT array logic internally.

    - Verifying an array manipulating program against its specification given as pre-post conditions and loop invariants (for program with loops).
    - Localizing the location of a bug when verification fails.
    - Synthesizing a fix for the bug when there is fix available **by manipulating the array access indices**. We will allow the array access index to be replaced by any index (or index plus a constant) in the program.
      In this project we will focus on the bugs due to array access operations.

3. Implementation of our method within the Rosette-Racket solver-aided programming tool/language framework.

4. Experiment our implementation on a targeted class of benchmark examples.

# 4 Expected Results

This is a work under progress being done as a project for an SMT course. By the time of SMT School we expect to have an implementation of the tool in Rosette with results of running our tool on a set of benchmark array manipulating programs.

# References

[1] E. Torlak and R. Bodik. Growing solver-aided languages with Rosette. In Onward!, 2013.

[2] Manu Jose and Rupak Majumdar, Cause Clue Clauses: Error Localization Using Maximum Satisfiablity, ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI), June 2011, San Jose, California, USA.

[3] J. Christ and J. Hoenicke, Weakly Equivalent Arrays, SMT 2014