# STM Project Proposal

Hitarth
Bhishmaraj

MSc CS Students, CMI

October 24, 2019

# 1 Introduction

Solver-aided programming language/framework, such as Rosette [?], 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 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. Hitarth: Find out which paper Prof. Srivas want me to refer in place of "?" ?

Srivas: Please check moodle under arraydecproc for the paper; you guys seem never bother to check anything I upload on Moodle!!

We simplify our problem by restricting ourselves to decidable fragments of the arrays theory (?), and we also restrict the class of programs that we will allow for analysis. .

# 2 Problem: Verification, Debugging and Synthesis in straight line array programs

Consider a simple program as given below:

```
a: array 0..99 of integer;
i: integer;

for i:=0 to 99 do
        assert($\forall x \in \mathbb{N}_0.\ x < i \implies a[x] = 0$);
        a[i]:=0;
        assert($\forall x \in \mathbb{N}_0.\ x \leq i \implies a[x] = 0$);
done;
assert($\forall x \in \mathbb{N}_0.\ x \leq 99 \implies a[x] = 0$);
```

The postcondition for this program is as follows:

$$\begin{aligned}&(\forall x \in \mathbb{N}. x < i \implies a[x] = 0)\\ &\wedge a' = a\{i \leftarrow 0\}\\ &\implies (\forall x \in \mathbb{N}. x \leq i \implies a'[x] = 0)\end{aligned} \tag{1}$$

Here $a$ and $a'$ represent the array before and after the execution of the program respectively. Given the pre/post conditions, we can ask for the correctness of the program. As this program is correct, it will satisfy the postcondition. If the program was not correct, then with the help of Rosette we can find out the cause of error and localize the fault.

Hitarth: Do I need to write an example of faulty program and show how we want tol get the fault localized (for demonstration purposes)?

Usually while writing the programs involving arrays, we might make an error while accessing the arrays. This is exactly what we will be focussing on while synthesizing. We shall assume that the error is only on the accessing operation, and with the help of Rosette we will try to fix that.

**Methodology:**

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 exists a fix within a restricted class of possible fixes.
     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.

# 3 Approach

We will firstly define a basic language which only deals with arrays of integers and provides basic functionalities like accessing and storing in an array. Then we will embed this language with basic programming constructs for arrays in Racket/Rosette.

Rosette provides us with various tool to verify, debug and synthesize once we write the interpreter in it. We will then explore how we can automate this process and auto fix the bugs related to array accessing.

We will show with examples how we can use this for debugging and synthesis in such programs.

We also plan to create a seperate GUI application which can provide an easy to use interface to the user to get these functionalities.

# 4 Expected Results

We expect to have a system which can take simple array programs and help the user with verification, debugging and synthesis provided they give either pre/post conditions along with the loop invariants.
Almost all the programs use arrays in one way or another, but most of the useful programs involving arrays also have loops. At least for now, our plan is not to indulge with loops. We might consider the loops if we are successful in the first phase of our project.

# 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.