

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 its 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. [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 a decidable fragment of arrays theory [?], and we also restrict the class of programs that we target for analysis. .

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

[Srivas: As I suggested yesterday, use a simple \(buggy\) array program with multiple assignment statements possibly conditional to clearly illustrate the three tasks of verification, bug-localization and bug-fixing.](#)

For the demonstration of problem, we shall give an example. Consider a simple program which swaps the value at i and j index of an array a if they are not in ascending order. We are writing the program in C-like syntax for demonstration only, in practice, the program will be written in the grammar defined by our language with the pre/post conditions.

We have introduced an **error** in the program by making $temp = a[i]$ instead of $temp = a[j]$, which was required for the successful working of the program.

```
int [] a;  
int i, j;  
@Pre : assert( $a_i = a[i] \wedge a_j = a[j]$ )
```

```

if (a[i] > a[j]) {
  temp = a[i];
  a[i]   = a[j];
  a[j]   = temp;
}
else continue;

```

@Post : **assert**($a_i > a_j \implies a[i] = a_j \wedge a[j] = a_i$)

@Post : **assert**($\neg(a_i > a_j) \implies a[i] = a_i \wedge a[j] = a_j$)

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 .

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

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.

3 Approach

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

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 Satisfiability, ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI), June 2011, San Jose, California, USA.