System LAV and Its Applications

Milena Vujošević-Janičić

Progress in Decision Procedures:
From Formalizations to Applications

Belgrade,
March 30, 2013.

Overview

Milena Vujošević-Janičić, Viktor Kuncak

Development and Evaluation of LAV: an SMT-Based Error Finding Platform.
Verified Software: Theories, Tools, Experiments.
Lecture Notes in Computer Science, Volume 7152, Springer 2012,
ISBN 978-3-642-27704-7.

Milena Vujošević-Janičić, Mladen Nikolić, Dušan Tošić, Viktor Kuncak

Software Verification and Graph Similarity for Automated Evaluation of Students' Assignments.
Information and Software Technology. Elsevier, 2013
DOI: 10.1016/j.infsof.2012.12.005

People

prof. Dušan Tošić
Faculty of Mathematics, Belgrade

prof. Viktor Kuncak
EPFL
Switzerland

Mladen Nikolić
Faculty of Mathematics, Belgrade

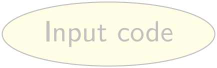
LAV

LAV

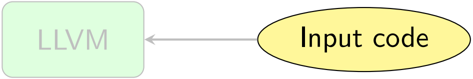
LAV is a tool for statically verifying program assertions and locating bugs such as buffer overflows, pointer errors and division by zero.

- LAV is publicly available and open-source
  http://argo.matf.bg.ac.rs/?content=lav
- Experimental evaluation shows that LAV is competitive with related tools based on symbolic execution and bounded model checking (KLEE, CBMC and ESBMC)

LAV

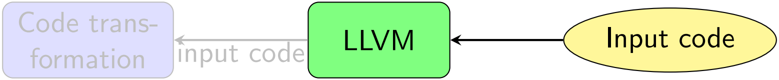


LAV



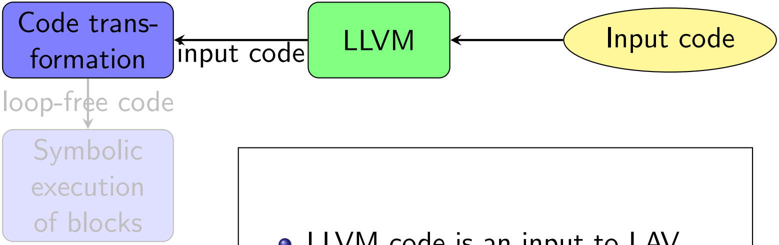
- LAV is primarily developed for programs in C
- Thanks to LLVM, LAV can be used for other procedural languages
- Input program is transformed to LLVM code

LAV



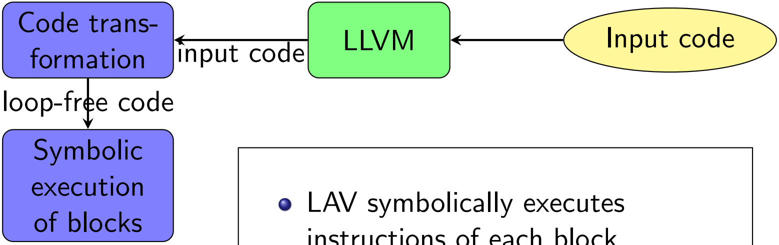
- LLVM is a rich compiler framework
- Each LLVM function consists of blocks of instructions which are interconected
- LLVM uses simple RISC-like instructions

LAV



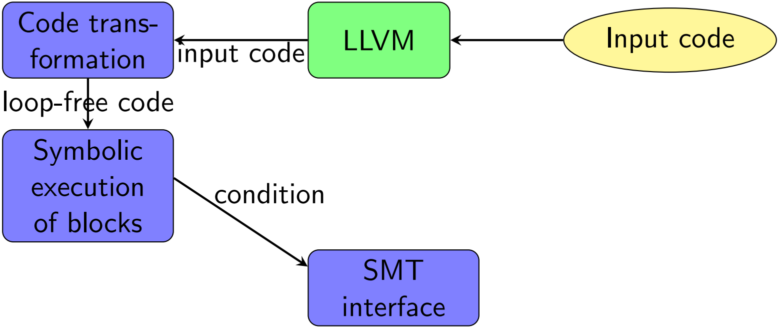
- LLVM code is an input to LAV
- LAV unrolls loops to get loop-free program
- LAV can unroll loops in two ways, by using under-approximation or over-approximation

LAV

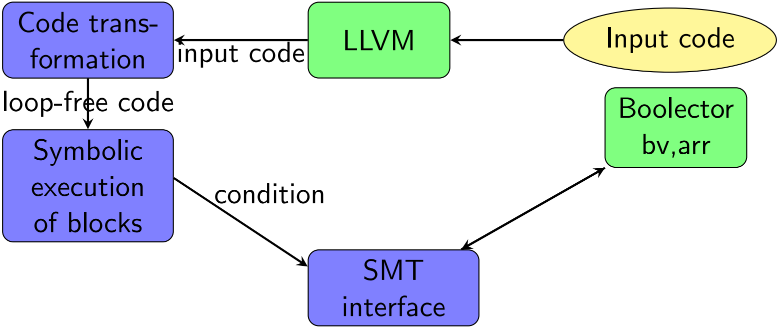


- LAV symbolically executes instructions of each block
- For safety critical commands, LAV generates safety conditions
- LAV checks these conditions in different contexts
- LAV communicates with an SMT solver through an SMT interface

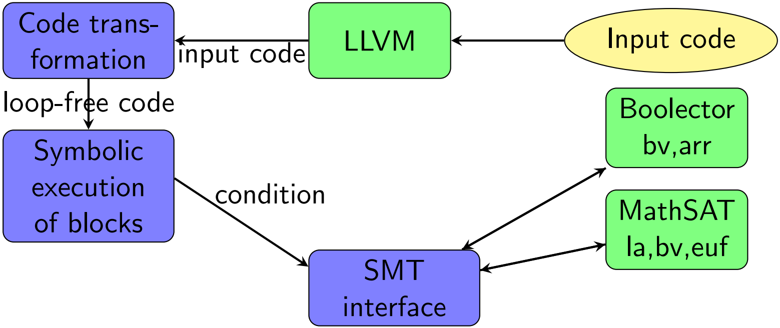
LAV


Code trans-formation
input code
LLVM
Input code
loop-free code
Symbolic execution of blocks
condition
SMT interface

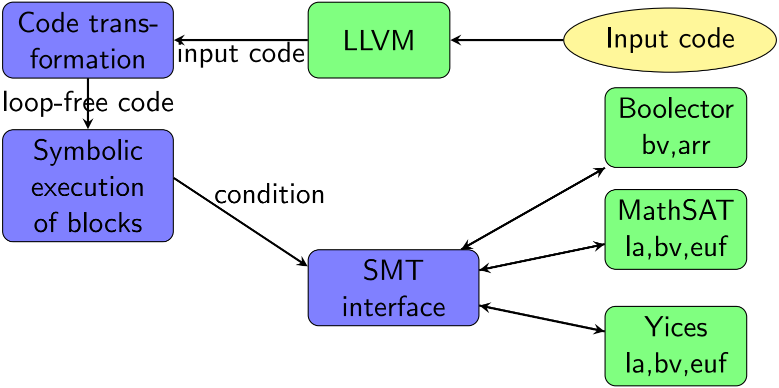
LAV


Code transformation
input code
LLVM
Input code
loop-free code
Boolector
bv,arr
Symbolic execution of blocks
condition
SMT interface

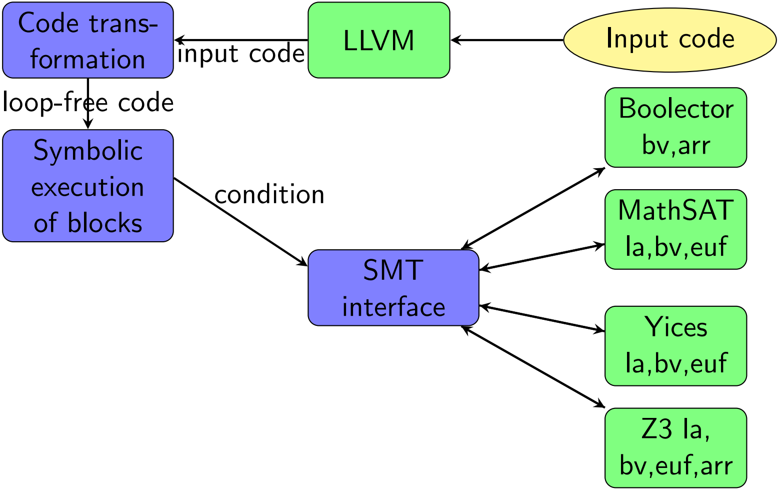
LAV


Code trans-
formation
input code
LLVM
Input code
loop-free code
Boolector
bv,arr
Symbolic
execution
of blocks
condition
MathSAT
la,bv,euf
SMT
interface

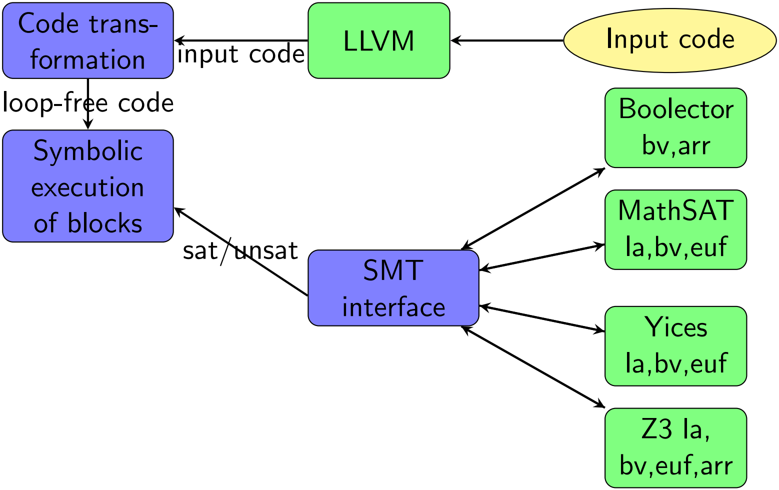
LAV


Code trans-formation
LLVM
Input code
input code
loop-free code
Symbolic execution of blocks
condition
SMT interface
Boolector bv,arr
MathSAT la,bv,euf
Yices la,bv,euf

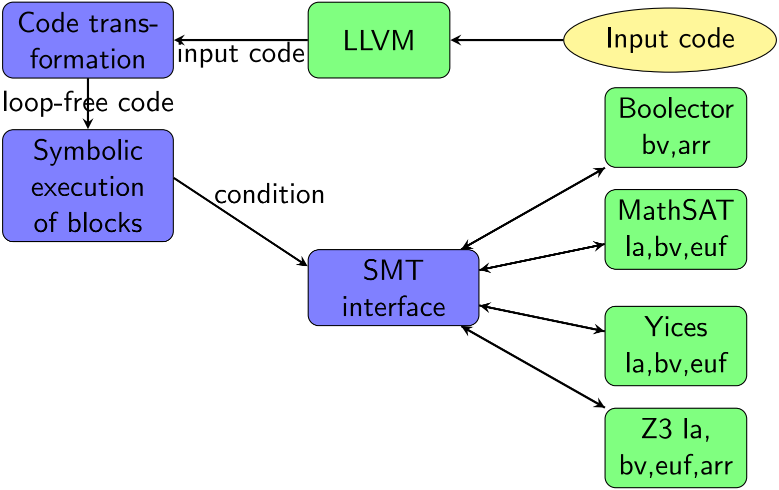
LAV


Code transformation
input code
LLVM
Input code
loop-free code
Symbolic execution of blocks
condition
SMT interface
Boolector bv,arr
MathSAT la,bv,euf
Yices la,bv,euf
Z3 la, bv,euf,arr

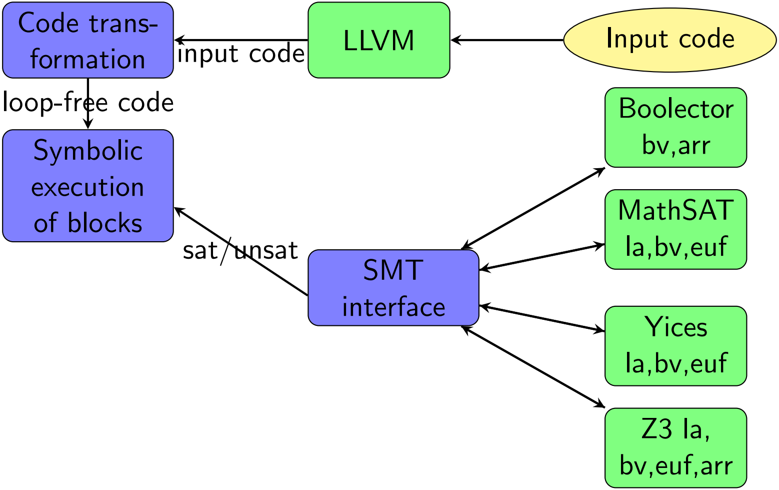
LAV


Code transformation
input code
LLVM
Input code
loop-free code
Symbolic execution of blocks
Boolector bv,arr
MathSAT la,bv,euf
sat/unsat
SMT interface
Yices la,bv,euf
Z3 la, bv,euf,arr

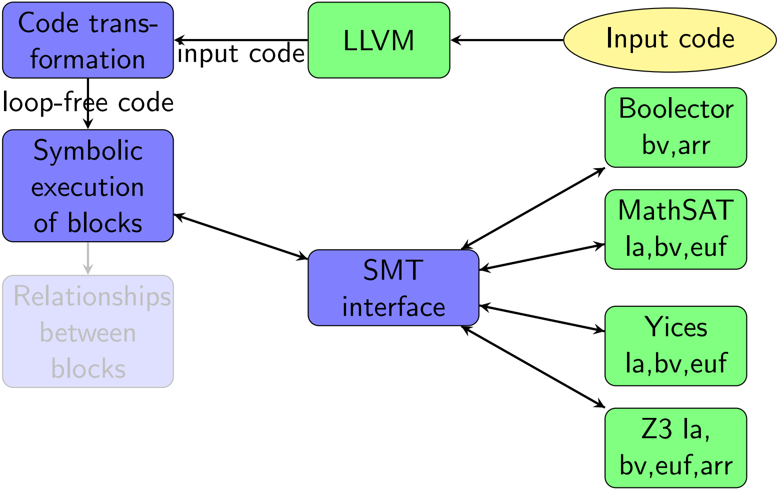
LAV


Code trans-formation
input code
LLVM
Input code
loop-free code
Boolector bv,arr
Symbolic execution of blocks
condition
MathSAT la,bv,euf
SMT interface
Yices la,bv,euf
Z3 la, bv,euf,arr

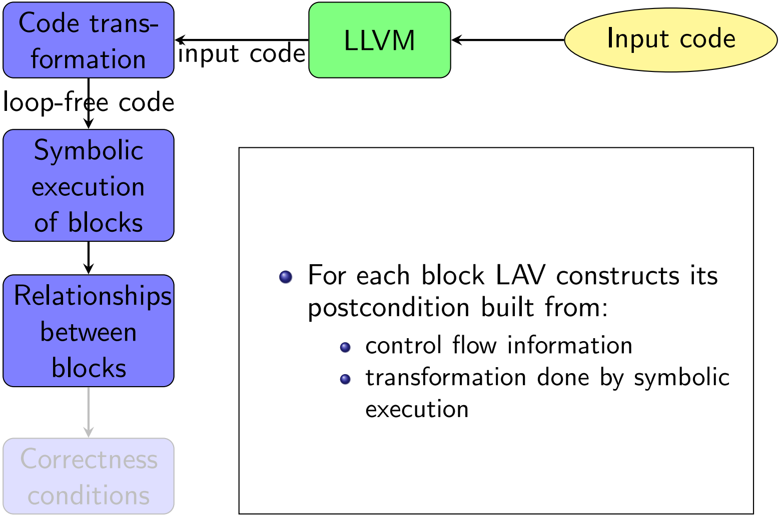
LAV


Code trans-formation
input code
LLVM
Input code
loop-free code
Boolector bv,arr
Symbolic execution of blocks
MathSAT la,bv,euf
sat/unsat
SMT interface
Yices la,bv,euf
Z3 la, bv,euf,arr

LAV

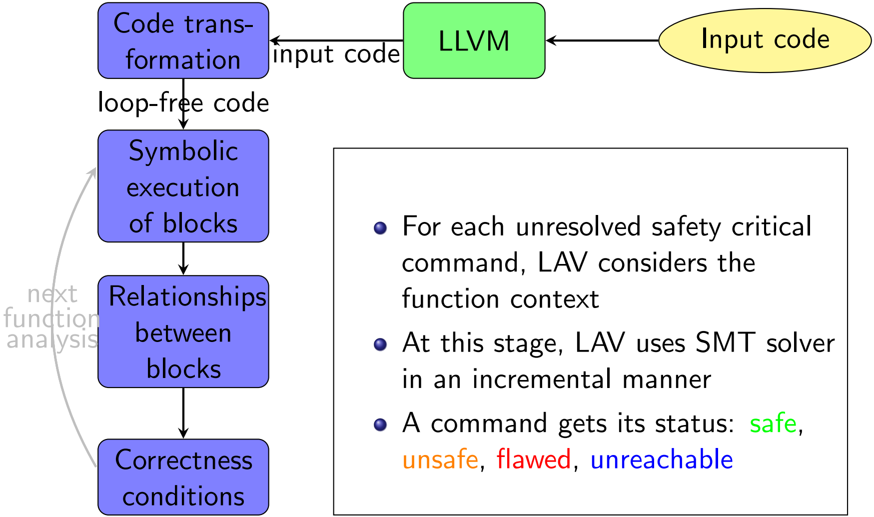

Code trans-formation
input code
LLVM
Input code
loop-free code
Symbolic execution of blocks
Relationships between blocks
SMT interface
Boolector bv,arr
MathSAT la,bv,euf
Yices la,bv,euf
Z3 la, bv,euf,arr

LAV



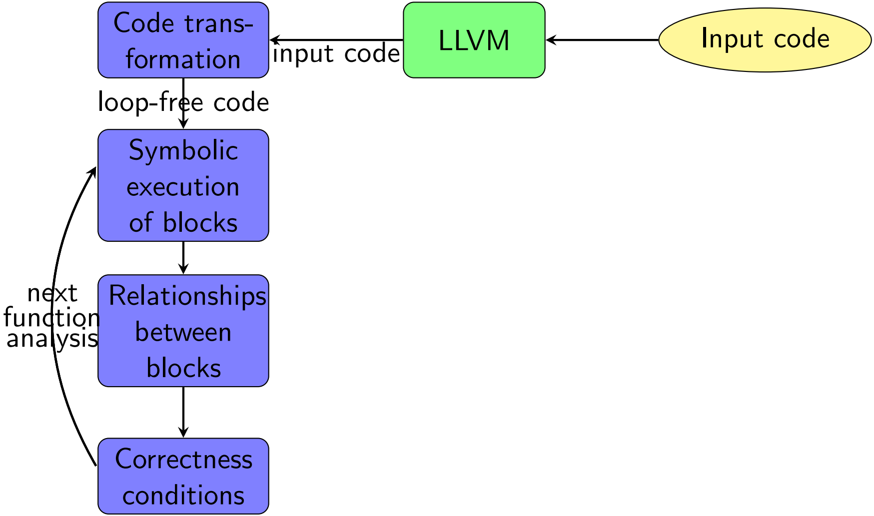
- For each block LAV constructs its postcondition built from:
  - control flow information
  - transformation done by symbolic execution

LAV

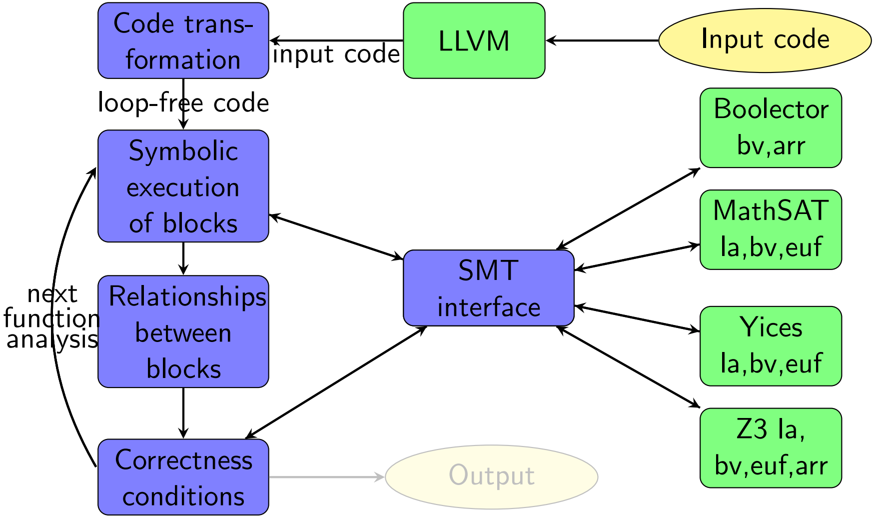


- For each unresolved safety critical command, LAV considers the function context
- At this stage, LAV uses SMT solver in an incremental manner
- A command gets its status: safe, unsafe, flawed, unreachable

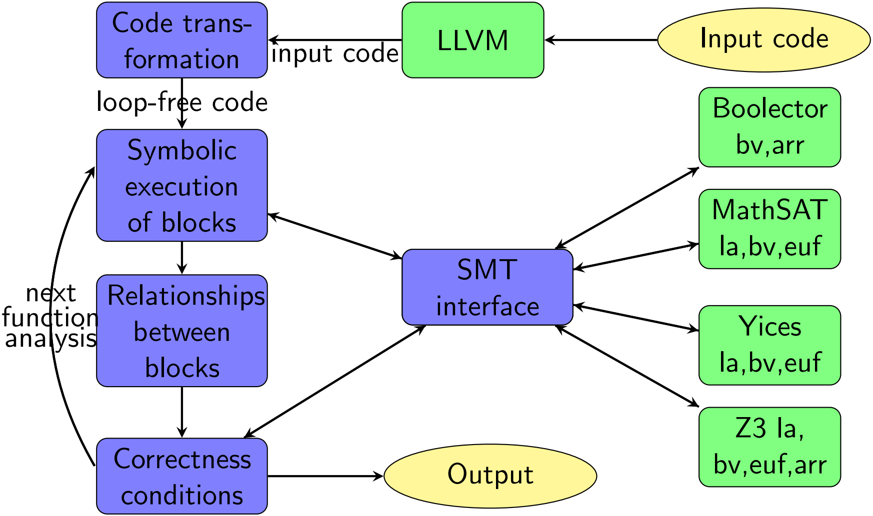
LAV


Input code
LLVM
input code
Code transformation
loop-free code
Symbolic execution of blocks
Relationships between blocks
Correctness conditions
next function analysis

LAV


Code transformation
LLVM
Input code
input code
loop-free code
Symbolic execution of blocks
Relationships between blocks
Correctness conditions
next function analysis
SMT interface
Boolector bv,arr
MathSAT la,bv,euf
Yices la,bv,euf
Z3 la, bv,euf,arr
Output

LAV


Code trans-formation
input code
LLVM
Input code
loop-free code
Boolector bv,arr
Symbolic execution of blocks
MathSAT la,bv,euf
SMT interface
next function analysis
Relationships between blocks
Yices la,bv,euf
Z3 la, bv,euf,arr
Correctness conditions
Output

Output

Output format

For a command that is not safe, LAV outputs the line number, the kind of the error, a program trace that introduces the error, and values of variables along this trace.

```
1: #include <stdio.h>
2: #define MAX 10
3: int main()
4: {
5: int arr[MAX],d,n,n_copy,max;
6: get_array(arr, MAX);
7: scanf("%d", &n);
8: n_copy = n;
9: max = n%10;
10:while(n){
11:   d = n%10;
12:   if(max<d)
13:       max=d;
14:   n = n/10;
15:}
16:if(arr[max]>n_copy)
17:   printf("it is bigger\n");
18:else printf("it is not bigger\n");
19:}
```

```
| verification failed:
| line 16: UNSAFE
| function: main, error: buffer_overflow
|
| 16: d = -8, max = -8, n = 0, n_copy = -899
| 10: d = -8, max = -8, n = 0, n_copy = -899
| 14: d = -8, max = -8, n = 0, n_copy = -899
| 13: d = -8, max = -8, n = -8, n_copy = -899
| 12: d = -8, max = -9, n = -8, n_copy = -899
| 10: d = -9, max = -9, n = -8, n_copy = -899
| 14: d = -9, max = -9, n = -8, n_copy = -899
| 12: d = -9, max = -9, n = -89, n_copy = -899
| 10: d = -9, max = -9, n = -89, n_copy = -899
| 14: d = -9, max = -9, n = -89, n_copy = -899
| 12: d = -9, max = -9, n = -899, n_copy = -899
| 10: d = 0, max = -9, n = -899, n_copy = -899
```

LAV in Education

- Verification tools are usually used for safety critical programs
- New domain: Education
- LAV analyzed corpus of 157 students' programs (tests written at introductory programming course)
- LAV found **423** errors!

LAV in Education

- Verification tools are usually used for safety critical programs
- New domain: Education
- LAV analyzed corpus of 157 students' programs (tests written at introductory programming course)
- LAV found **423** errors!

Verification in education

Can a verification tool help in automated evaluation of students' programs?

LAV in Education

- Verification tools are usually used for safety critical programs
- New domain: Education
- LAV analyzed corpus of 157 students' programs (tests written at introductory programming course)
- LAV found **423** errors!

Verification in education

Can a verification tool help in automated evaluation of students' programs?

Automated evaluation

- Beneficial for both teachers (automated grading) and students (immediate feedback)
  - Can a verification tool help teachers in grading assignments?
  - Can a verification tool help students to learn programming?
- Important for introductory programming courses – the biggest number of students
- With a growing number of on-line courses, this automation becomes even more significant

Automated evaluation

- Beneficial for both teachers (automated grading) and students (immediate feedback)
  - Can a verification tool help teachers in grading assignments?
  - Can a verification tool help students to learn programming?
- Important for introductory programming courses – the biggest number of students
- With a growing number of on-line courses, this automation becomes even more significant

Testing vs. verification

- Automated evaluation based on automated testing — does the program give the desired outputs for specific inputs?
- Standard problem: Testing cannot reveal all bugs
- Are there bugs that can be discovered by a verification tool in a program that successfully passed testing?
- 266 programs (solutions of 15 different problems) successfully passed testing

Testing vs. verification

- Automated evaluation based on automated testing — does the program give the desired outputs for specific inputs?
- Standard problem: Testing cannot reveal all bugs
- Are there bugs that can be discovered by a verification tool in a program that successfully passed testing?
- 266 programs (solutions of 15 different problems) successfully passed testing — LAV found **35** errors

Testing vs. verification

- Automated evaluation based on automated testing — does the program give the desired outputs for specific inputs?
- Standard problem: Testing cannot reveal all bugs
- Are there bugs that can be discovered by a verification tool in a program that successfully passed testing?
- 266 programs (solutions of 15 different problems) successfully passed testing — LAV found **35** errors

Testing vs. verification

- Randomly generated test-cases (fuzzing) discovered only 12 of these bugs and took significantly more time
- Important issue (concerning feedback to students):
  - If a program crashes, testing cannot give an explanation why it crashed
  - Verification tools can give explanations

Similarity measure

- Functional correctness is not the only important aspect in automated evaluation
- Requirements concerning the design of a solution cannot be addressed by testing and verification
- To address the design, we used similarity measure between CFGs of expected solution and student's solution:

  Mladen Nikolić, Measuring Similarity of Graph Nodes by Neighbor Matching, *Intelligent Data Analysis, (2013).*

- Open-source, publicly available from LAV's page

Similarity measure

- Functional correctness is not the only important aspect in automated evaluation
- Requirements concerning the design of a solution cannot be addressed by testing and verification
- To address the design, we used similarity measure between CFGs of expected solution and student's solution:

Mladen Nikolić, Measuring Similarity of Graph Nodes by Neighbor Matching, *Intelligent Data Analysis, (2013).*

- Open-source, publicly available from LAV's page

Automated grading

Predicting a grade

Linear model based on testing (t), verification (v) and CFG similarity (s)

$$grade = c_1 \cdot t + c_2 \cdot v + c_3 \cdot s$$

- Coefficients c_i are determined based on a set of instructor graded solutions
- The model obtained high correlation between predicted and instructor provided grades (84%)
- Better results compared to the model without verification component and to the model without similarity component
- All obtained results statistically significant

Conclusions

LAV

LAV is an SMT-Based error finding platform

- Uses symbolic execution and elements of BMC
- Competitive to related tools

Verification in education

Verification can add new quality to automated evaluation of students' assignments:

- Feedback for students
- Automated grading for teachers

Future work

LAV

Further improvements of LAV

Application of LAV

Integrating automated evaluation into a web-based system

The end

Thank you for your attention!