Summary
We present the static LLVM bytecode analyzer LGA capable of automatic bug finding and program verification. Analysis relies on the SMT and LP solvers, and scales well with input program size in the practical case.

Motivation
- In today’s world more and more safety critical systems become computerized.
- Bugs in such software systems incur massive damage.
- Traditional methods of checking correctness, such as unit testing and code reviews are no longer sufficient.
- Static program analysis approach can be used for software verification.
- One of famous disastrous bugs is the launch of Ariane rocket in 1996 (Fig. 1).
- Static analysis can be used in program verification — proving the absence of such bugs.

Example of the verification for the simple “Pyramid” program

```
int main()
{
    int n; // node <start>
    scanf("%d", &n);
    int no_columns = 2 * n - 1;
    for (int i=0; i<n; i++) { // node <main_loop>
        int no_stars = 2 * i + 1;
        int no_spaces = n - i - 1;
        for (int s=0; s<no_spaces; s++) {
            printf(" "); // node <space_loop>
        }
        printf("*"); // node <star_loop>
    }
    printf("\n");
}
```

Figure 3: “Pyramid” program C code

```
Figure 4: Program output for n = 3
```

Conclusion
- We’ve developed LGA static analyzer
- Algorithm: max-strategy iteration in template constraint domain combined with path-focusing via model-checking (Gawlitza and Monniaux [2011])
- Associated decision problem complexity: \( \Pi_2 \)
- Clever implementation required to make it scalable for the typical program.
- LGA accepts LLVM bytecode as an input, and hence can analyze most of the statically-typed languages
- Analysis results can be used to find bugs or to verify correctness.

Background
Static Analysis
- Static analysis: analysis of software without running it.
- Complete static analysis of a program is equivalent to a halting problem and is undecidable.
- Some information about the program can be still obtained.
- The method used for LGA produces sound, but incomplete result.
- E. g. if LGA says that program is safe, it is indeed safe.
- But if LGA says it is unsafe, it might not be the case.

Max-Strategy Iteration
- Max-strategy iteration: game-theoretic approach which was recently utilized for static analysis (Gawlitza and Helmut [2007]).
- Improves precision and performance of traditional Kleene iteration, guarantees termination.
- Guarantees to obtain the best possible solution in the given domain
- Limitation: tracks only affine statements
- Path focusing: merging multiple edges into one and using an SMT solver to “navigate” inside such edges (Gonnord and Monniaux [2011])

Terminology
- LLVM, or low-level virtual machine is a compiler infrastructure with front-ends for many popular statically-typed languages (C, C++, Objective-C, etc.).
- LGA (Satisfiability Modulo Theories) solver — solver which is capable of efficiently solving NP-complete SAT problem with linear equations in an average case.
- LP — solver which is capable of solving a linear maximization problem efficiently

Implementation
- Pre-processing
  - Control flow graph
- Policy Improvement
  - SMT solver
- Value determination
  - LP solver
- Output
  - Visualizer

Figure 2: LGA analysis pipeline

Analysis output
- Program or user annotates lines of interest in program with expressions, e. g. \( x+y = 3z \)
- LGA computes lowest bound for this expression
- Useful in checking the absence of array-out-of-bounds, zero-divisions, overflows, etc.

Results
- On a running example of Gawlitza and Monniaux [2011], LGA obtains the 30x speed-up vs. proof-of-concept implementation by authors

Example
- The program in Fig. 3 draws an ASCII pyramid shown in Fig. 4.
- Control-flow graph representation: Fig. 5
- Verification condition: no_columns = no_stars + 2 * no_spaces
- LGA verifies this invariant in 0.005 seconds

References