Static Analysis: Automated Bug Hunting and Beyond

Michael Schwarz Julian Erhard
{m.schwarz, julian.erhard}@tum.de

Chair for Formal Languages, Compiler Construction, Software Construction
Department of Informatics
Technical University of Munich

Summer Term 2021

Writing programs is hard.

Writing correct programs is very hard.

Testing

- Widely successful
- Can be automated to some extent
- Can only show that there are bugs, not their absence

Machine-verified proof (e.g. Isabelle)

- Can show bugs & their absence
- A highly manual process requiring highly trained people
- Problem with proof and implementation diverging

Static Analysis

- Fully automated
- Can show absence of certain classes of bugs
- Runs directly on the input program
- Abstract Interpretation, Model Checking, ...

Static Analysis

- Fully automated
- Can show absence of certain classes of bugs
- Runs directly on the input program
- Abstract Interpretation, Model Checking, ...

Abstract Interpretation

- Widely used both in Academia & Industry
- Can scale to huge industry-scale codebases
- The technique covered in Program Optimization Course (IN2053)

Goblint

- Analysis of multi-threaded, real-world C
- Efficient solvers for computation of fixpoints
- `https://goblint.in.tum.de`

Topics

- Integer Domains
    - Congruences
    - Octagons
    - ...
- Undefined Behavior
    - Null-Pointer-Dereference
    - Access-Out-Of-Bounds
    - ...

Example 1

Program correctness may depend on relational information between variables:

```
void main() {
    int n = rand(); // Initialize to random value
    if(n<0){
        return;
    }
    int i = 0;
    for(; i<n; i++){
        printf("foo\n");
    }
    if(i != n)
        crash(); // Something went horribly wrong
}
```

Example 1

Program correctness may depend on relational information between variables:

```
void main() {
    int n = rand(); // Initialize to random value
    if(n<0){
        return;
    }
    int i = 0;
    for(; i<n; i++){
        printf("foo\n");
    }
    if(i != n)
        crash(); // Something went horribly wrong
}
```

$\longrightarrow$ Use Octagon domain for relational information

Octagon Domain

- Store conjunction of constraints of the form $\pm X \pm Y \leq c$ where X and Y are program variables, and c is an integer.
- More precise information than intervals, but also more computationally expensive

Example 2

```
#include<stdlib.h>
#define LENGTH 10
int main(){
    int *values = malloc(LENGTH *  sizeof(int));

    int i;
    for(i=0; i<LENGTH; i++){
        values[i] = i;
    }

    for(i=0; i<LENGTH; i++){
        values[i] = values[i]+values[(i%LENGTH)+1];
    }
    free(values);
}
```

Example 2

```c
#include<stdlib.h>
#define LENGTH 10
int main(){
    int *values = malloc(LENGTH *  sizeof(int));

    int i;
    for(i=0; i<LENGTH; i++){
        values[i] = i;
    }

    for(i=0; i<LENGTH; i++){
        values[i] = values[i]+values[(i%LENGTH)+1];
    }
    free(values);
}
```

The `values` array is accessed outside its bounds!

Benefits

- Deepen your understanding of
  - The Semantics of C and typical programming errors
  - Static Analysis by Abstract Interpretation
- Train your functional programming skills
- Give some insights into developing a research prototype

Format

- Teams of 2-4 students
- Course will take place throughout the semester
- (Bi-)weekly meetings with (one of) us
- Presentation at the end (one day, all groups)
  - Attendance & Active Participation mandatory(!)

Requirements

- Program Optimization Course (IN2053) (or a similar course at another university)
- Knowledge of a functional programming language (we use OCaml)
- Be in your Master's (Advanced Bachelor's students welcome)

Questions?

Further Reading

International standard ISO / IEC 9899:1999 Programming languages C - technical corrigendum 3 - Committee Draft.
ISO, 2007.
URL: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf.

Antoine Miné.
The octagon abstract domain.
In Elizabeth Burd, Peter Aiken, and Rainer Koschke, editors, *Proceedings of the Eighth Working Conference on Reverse Engineering, WCRE'01, Stuttgart, Germany, October 2-5, 2001*, page 310. IEEE Computer Society, 2001.
URL: https://doi.org/10.1109/WCRE.2001.957836, doi:10.1109/WCRE.2001.957836.