Automatic Generation of Test Cases for Incremental Static Analysis

Jonas August

Munich, 08/22/2023

Outline

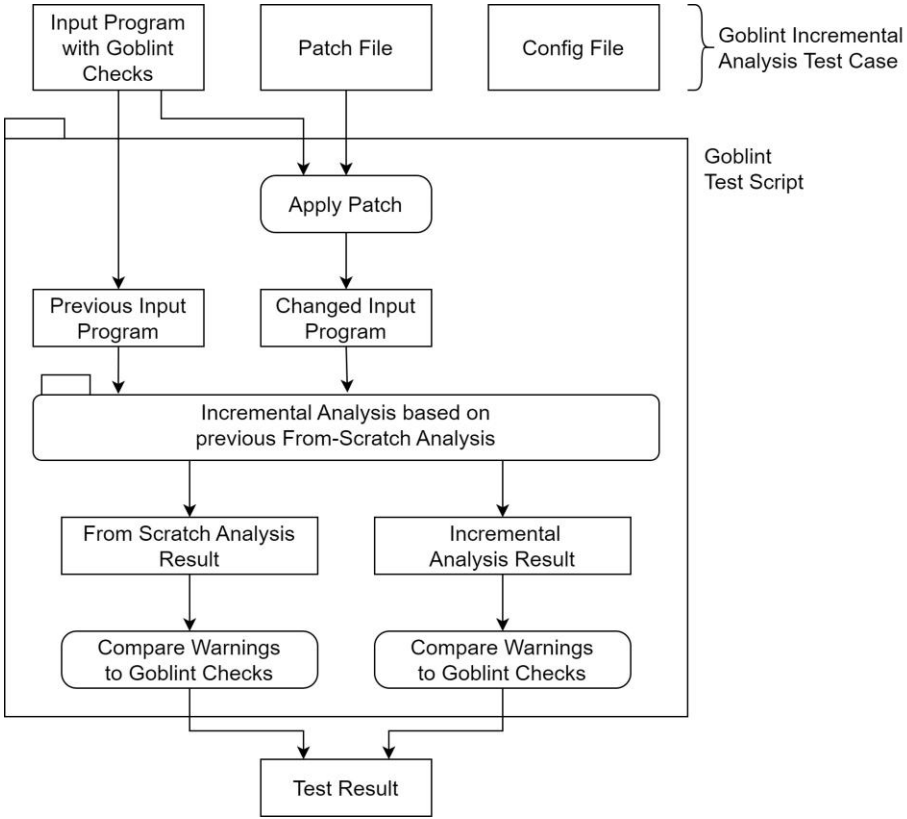
- Introduction and Motivation
- Testing the Incremental Analysis
- Automated Test Case Creation
- Implementation: Test Automation for Incremental Analysis (TAIA)
  - Syntax based Mutations
  - AI based Mutations
- Evaluation
- Conclusion

Introduction and Motivation

- Static analysis tools are important in software development
- Incremental analysis as performance improvement
- Goblint as example for a static analysis tool
  - Test cases for incremental analysis created manually
  - Only a few test cases available
- We show that the test case creation can be automated and streamlined

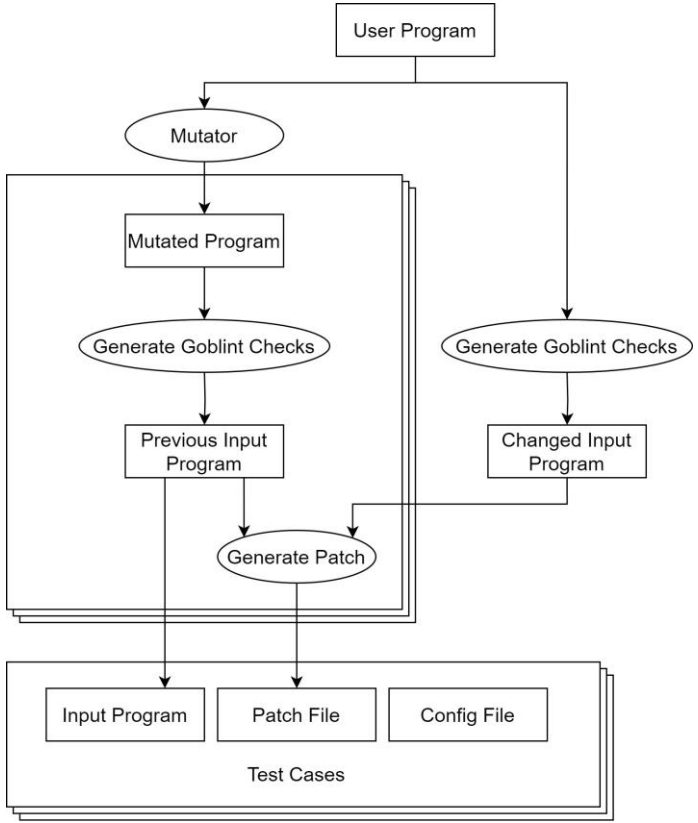
Testing the Incremental Analysis

- Goblint Checks encode expected result
  __goblint_check(x == 42); // SUCCESS
- Test cases contain two versions of a program
  - Previous Input Program
  - Changed Input Program
- Tester compares warnings with Goblint Checks



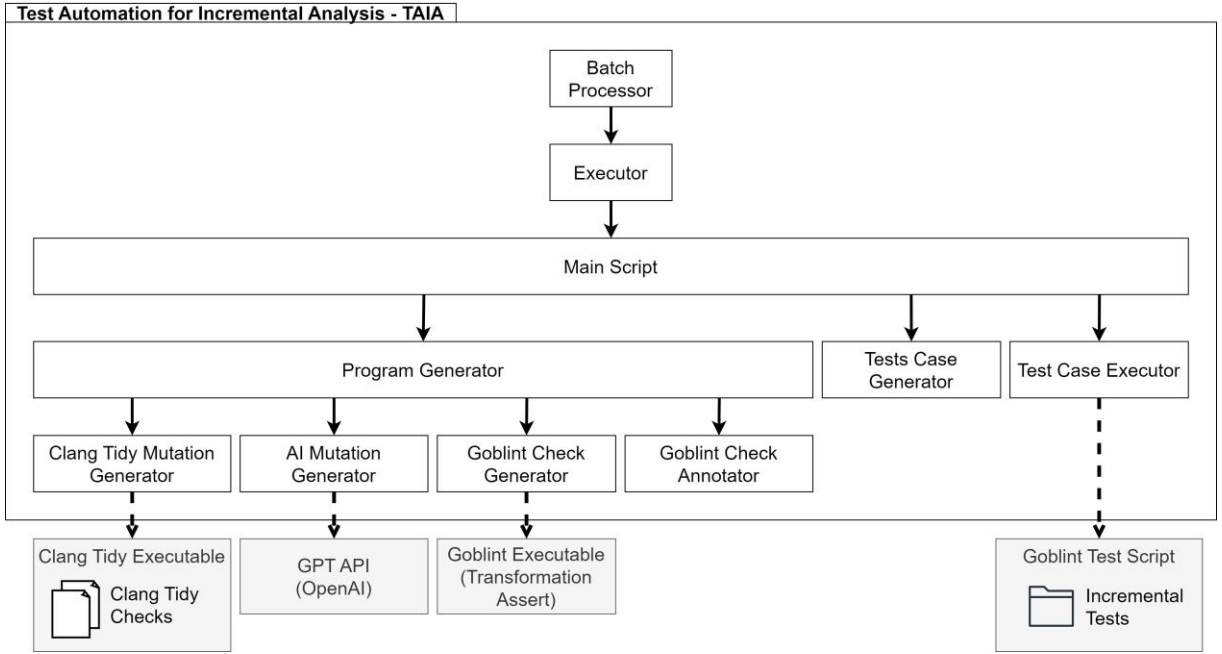
Automated Test Case Creation

- Mutator generates mutated program versions automatically
- Goblint Checks generated with Transformation Assert
- Multiple test cases based on one single user program



Implementation:
Test Automation for Incremental Analysis (TAIA)

- Layered architecture pattern
- Automatic Generation of test cases
  - Syntax based mutations (Clang-Tidy)
  - Artificial intelligence based mutations (GPT API)
  - Goblint Checks
  - Test case structure
- Direct execution of Goblint Test Script



Clang-Tidy Mutation Generator (Syntax based)

- Clang-Tidy Checks define rules to match and transform patterns in the syntax
- We created seven different Clang-Tidy Checks that resemble real-world program changes
- Mutated program is semantically less meaningful and therefore used as previous input program

Clang-Tidy Check (Mutation Type)	Resembles	Previous Input Program / Mutated Program	Changed Input Program
Remove Function Body	Implement a function	int function() {**return 0;**}	int function() {**int x = 1; return x;**}
Remove Thread	Implement parallelism	int t = **function_wrapped()**;	int t = **pthread_create(&function)**;
Constant Replacement	Increase number of threads	int x = **1**;	int x = **42**;
Remove If Statement	Add missing error handling	**;**	**if (pointer == NULL) {}**
Unary Operator Inversion	Fix boolean logic bug	if (**!(**a<b**)**) {}	if (a<b) {}
Logical Connector Replacement	Fix boolean logic bug	if (a **||** b)	if (a **&&** b)
Relational Operator Replacement	Fix off-by-one errors	if (a **<=** b)	if (a **<** b)

AI Mutation Generator (AI based)

- Generative AI can understand and modify code
- We use the GPT API from OpenAI
- Prompt used to specify and request mutations
  - Sets context
  - Excludes mutations created by Clang-Tidy Checks
  - Requests snippet that can be inserted into source code

You are a developer for C, helping me with my following question. […] My question is how a previous version of this code could have looked like before some typical code changes have been done by developers. Please generate me such a previous version!

The code you generate should be a self-contained snippet that could directly replace the provided excerpt in the initial, complete program. […]

Evaluation of Mutators

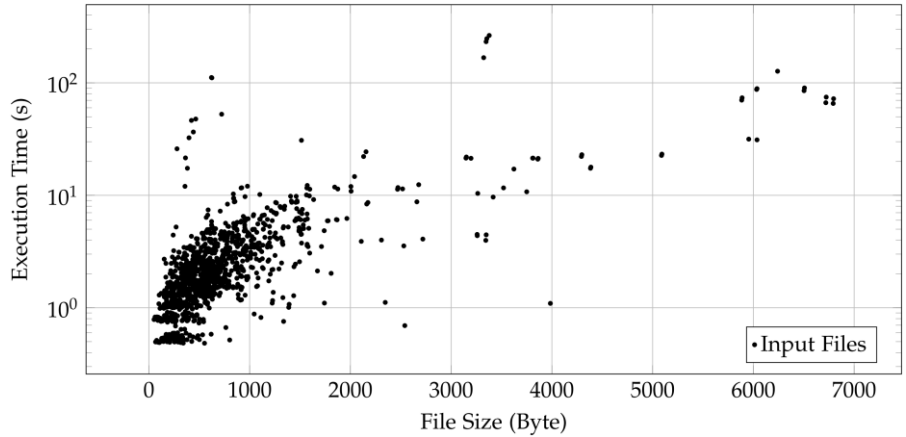
- Input files from the non-incremental regression tests of Goblint

	Clang-Tidy Mutation Generator	AI Mutation Generator
Specification of Mutations	Defined Rules (AST)	Statistical Model (Tokens)
Input Files	1 219	50
Generated Mutations per input file	x9.01 (10 980)	x1 (50)
Successful Mutations (Test Cases)	**99.35%** (10 710)	**52.00%** (26)
Ø Execution Time per input file (in sec)	0.2	11.5
Deterministic	✓ reproduceable	✗ varies for each Request
Costs for all Requests	Free of Charge	0.09$

Evaluation of Execution Time

- 1219 input files from the non-incremental regression tests of Goblint
  - Clang-Tidy Mutation Generator
  - 10710 generated test cases
  - 1h:51min overall execution time
    - Ø 5.5s per input file
    - Ø 622ms per test case

Execution Time based on Input File Size



Average Execution Time based on Input File Size

Buckets of File Size (Byte)	≤ 1000	> 1000 ≤ 2000	> 2000 ≤ 3000	> 3000 ≤ 4000	> 4000 ≤ 5000	> 5000 ≤ 6000	> 6000 ≤ 7000
Files in Bucket (Count)	1001	160	18	21	4	5	10
Avg. Exec. Time (ms)	2602	5013	9667	54231	20083	44267	78873
therein Test Execution	1072	2187	4209	30737	8888	19584	34626
therein Check Gener.	1155	2104	4131	20568	8733	19641	35672
therein GCC	192	410	841	2317	1729	3747	6422
therein Clang-Tidy	165	285	4411	525	652	1126	1866
therein Test Gener.	8	18	35	72	72	154	273

Conclusion

- Automation and streamlining of test case creation is possible
- Significant increase in number of test cases
- Clang-Tidy Mutation Generator highly suitable
  - Quick, deterministic, reliable, and free of charge
  - Large number of generated test cases
- AI Mutation Generator can be used as inspiration
  - Slow, non-deterministic, error-prone, and not free of charge
  - Can be used as inspiration for new Clang-Tidy Checks
- Further mutation types can be added with minimal effort
- TAIA can be integrated into the development process of Goblint