Fully Automated Compiler Testing of a Reasoning Engine via Mutated Grammar Fuzzing

Yavuz Koroglu\textsuperscript{1} Franz Wotawa\textsuperscript{2}

\textsuperscript{1}Department of Computer Engineering, \textit{Bogazici University}, TURKEY
\textsuperscript{2}Institute for Software Technology, \textit{Graz University of Technology}, AUSTRIA

May 27, 2019, Montréal/CANADA
14th ACM/IEEE International Workshop on Automation of Software Test (AST)
41st ACM/IEEE International Conference on Software Engineering (ICSE)
Overview

1. Problem Overview

2. Our Reasoning Engine: ATMS
   - DEMO I
   - Compiler Errors

3. gFuzzer Overview
   - Small Example: bc
   - DEMO II
   - Mutation Operators

4. Evaluation
   - Test Generation Results
   - DEMO III

5. Future Work

6. Miscellaneous Information
Problem Overview

Main Problem

Develop a **fully-automated** (once started, requires no human intervention) testing tool that

1. Generates,
2. Executes, and
3. Evaluates
tests for a reasoning engine.

Reasoning Engine

A system that takes

<< Axioms, observations,

and returns

>> Logical consequences (diagnoses)
Our Reasoning Engine: ATMS

**Figure: Assumption-based Truth Maintenance System (ATMS) Overview**

**Necessary Condition for Reasoning**

Observations and axioms must be **correctly compiled**.

- **Compiler testing** is required.
Our Reasoning Engine: ATMS

Figure: Assumption-based Truth Maintenance System (ATMS) Overview

Example (Reasoning Model)

- observation
- assumption

- hitGas $\land$ Running $\rightarrow$ moving
- hitGas $\land$ Broken $\rightarrow$ notMoving
- moving $\land$ notMoving $\rightarrow$ ⊥

Example (Observation)

- hitGas
- moving
Our Reasoning Engine: ATMS

Figure: Assumption-based Truth Maintenance System (ATMS) Overview

Example (Diagnoses)

\[ D = \{\{\text{Running}\}\} \]
Proceed to DEMO
## Compiler Errors

<table>
<thead>
<tr>
<th>Actual</th>
<th>ATMS</th>
<th>Compile Failure</th>
<th>Consequence List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reject</td>
<td>Test Passed</td>
<td>Type I Error</td>
<td></td>
</tr>
<tr>
<td>Accept</td>
<td>Type II Error</td>
<td>Test Passed</td>
<td></td>
</tr>
</tbody>
</table>

### Type I Errors (False Positives)

Compiler **accepts** an **invalid input**.

### Type II Errors (False Negatives)

Compiler **rejects** a **valid input**.
gFuzzer Overview

- **Test Oracle**
  - Test Execution
  - Test Generation
  - ATMS
  - Grammar
  - gFuzzer
  - mgFuzzer
  - CYK
  - Recognizer

- **Outputs**
  - Agree?
  - Test Passed: YES
  - Test Failed: NO

Dotted Lines, gFuzzer
Dashed Lines, mgFuzzer
Solid Lines, both

**gFuzzer: Grammar Fuzzer**

- Takes a **context-free grammar** (ATMS Grammar in this case).
- **Generates** random sentences (Valid test inputs).
- **Executes** generated tests.
- ATMS must **always accept** (Checks only Type II Errors).
gFuzzer Overview

mgFuzzer: Mutated Grammar Fuzzer

- **Mutates** the original grammar.
- **Generates** random sentences (could be valid or invalid).
- **Executes** generated tests.
- Compares ATMS output with a **CYK recognizer**.
- Checks for both **Type I and Type II Errors**.
Small Example: bc

bc

bc is a UNIX tool that evaluates arithmetic expressions.

What can we do with gFuzzer?

- Exact grammar is NOT known,
  - Design a reasonable grammar and
  - Discover functionalities.
- Manual generation of grammars?
  - Infer a grammar from example test inputs.
  - Fuzz the inferred grammar.
Proceed to DEMO
Terminal Replacement (TR)

**Swaps** two non-equal terminals.

\[(\langle A \rangle ::= a, \langle B \rangle ::= b) \Rightarrow (\langle A \rangle ::= b, \langle B \rangle ::= a)\]

**Example**

\[(\langle \text{Digit} \rangle ::= 1, \langle \text{Op} \rangle ::= \rightarrow) \Rightarrow (\langle \text{Digit} \rangle ::= \rightarrow, \langle \text{Op} \rangle ::= 1)\]

**Note**

For simplicity, we assume **Chomsky Reduced Form (CRF)**. In CRF, there are only two types of rules:

1. \( \langle R \rangle ::= \langle Q \rangle \langle S \rangle \) and
2. \( \langle R \rangle ::= t \)
Mutation Operators II

DEletion (DE)

**Replaces** all expansions of a rule with empty string.

\[
\langle A \rangle \in G \Rightarrow \langle A \rangle ::= \varepsilon
\]

**Example**

\[
(\langle \text{NonEmptyList} \rangle ::= \langle \text{Element} \rangle \langle \text{Rest} \rangle ) \Rightarrow \langle \text{NonEmptyList} \rangle ::= \varepsilon
\]

**Note**

For simplicity, we assume **Chomsky Reduced Form (CRF)**. In CRF, there are only two types of rules:

1. \( \langle R \rangle ::= \langle Q \rangle \langle S \rangle \) and
2. \( \langle R \rangle ::= t \)
### DUplication (DU)

**Duplicates a rule.**

\[
\langle A \rangle ::= \langle B \rangle \langle C \rangle \Rightarrow (\langle A \rangle ::= \langle A' \rangle \langle A' \rangle, \langle A' \rangle ::= \langle B \rangle \langle C \rangle)
\]

### Example

\[
(\langle \text{Add} \rangle ::= \langle \text{Term} \rangle \langle \text{PlusTerm} \rangle) \Rightarrow (\langle \text{Add} \rangle ::= \langle \text{Add}' \rangle \langle \text{Add}' \rangle)
\]

### Note

For simplicity, we assume **Chomsky Reduced Form (CRF)**. In CRF, there are only two types of rules:

1. \langle R \rangle ::= \langle Q \rangle \langle S \rangle and
2. \langle R \rangle ::= t
EXchange (EX)

Swaps the order of non-terminals.

\[(<A> ::= <B> <C>) \Rightarrow (<A> ::= <C> <B>)\]

Example

\[(<\text{Add}> ::= <\text{Term} > <\text{PlusTerm}>) \Rightarrow (<\text{Add}> ::= <\text{PlusTerm} > <\text{Term}>)\]

Note

For simplicity, we assume Chomsky Reduced Form (CRF). In CRF, there are only two types of rules:

1. \(<R> ::= <Q> <S> \) and
2. \(<R> ::= t\)
Recursion Insertion (RI)

**Enables** infinite recursion on a random rule.

\[(\langle A \rangle \in G) \Rightarrow (\langle A \rangle ::= \langle A \rangle | \langle A \rangle \langle A \rangle)\]

**Example**

\[(\langle \text{False} \rangle ::= \bot) \Rightarrow (\langle \text{False} \rangle ::= \bot | \langle \text{False} \rangle \langle \text{False} \rangle)\]

**Note**

For simplicity, we assume **Chomsky Reduced Form (CRF)**. In CRF, there are only two types of rules:

1. \[\langle R \rangle ::= \langle Q \rangle \langle S \rangle\] and
2. \[\langle R \rangle ::= t\]
Terminal Insertion (TI)

**Inserts** a random terminal to a random rule.

\[(\langle A \rangle \in G) \Rightarrow (\langle A \rangle ::= \langle A \rangle \ x) \text{ or } (\langle A \rangle ::= x \langle A \rangle)\]

**Example**

\[(\langle \text{False} \rangle ::= \bot) \Rightarrow (\langle \text{False} \rangle ::= \neg \bot)\]

**Note**

For simplicity, we assume **Chomsky Reduced Form (CRF)**. In CRF, there are only two types of rules:

1. \(\langle R \rangle ::= \langle Q \rangle \langle S \rangle\) and
2. \(\langle R \rangle ::= t\)
Evaluation

Method

- Execute both *gFuzzer* and *mgFuzzer*, each **one week**.
- Measure **rule coverage**.
- Measure **code coverage**.
- Collect **failed tests**.

Rule (Production) Coverage Criterion

- **Each rule** in the grammar must be **expanded** at least once.
- **Expansion**: Replacing a rule in the grammar with its terms.
- Common in compiler testing.
### Test Generation Results

<table>
<thead>
<tr>
<th></th>
<th>Failed</th>
<th>Passed</th>
<th>Total</th>
<th>Rule (%)</th>
<th>Code (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>gFuzzer</em></td>
<td>0</td>
<td>1,490,388</td>
<td>1,490,388</td>
<td>100</td>
<td>67.6</td>
</tr>
<tr>
<td><em>mgFuzzer</em></td>
<td>2</td>
<td>1,024</td>
<td>1,026</td>
<td>100</td>
<td>75.9</td>
</tr>
<tr>
<td><strong>Both</strong></td>
<td>2</td>
<td>1,491,412</td>
<td>1,491,414</td>
<td>100</td>
<td>75.9</td>
</tr>
</tbody>
</table>

**Failed Tests**

<table>
<thead>
<tr>
<th>Test #1</th>
<th>x1,false x2() x3→x2. Assumption1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test #2</td>
<td>Assumption1, x2(false, x3)false. Assumption3. Assumption2. x1.</td>
</tr>
</tbody>
</table>

- *mgFuzzer* is considerably **slower** than *gFuzzer*.
- *mgFuzzer* clearly **outperforms** *gFuzzer* in code coverage.
- *mgFuzzer* finds an interesting error with **fewer tests**.
Proceed to DEMO
Future Work

Test the Reasoner

1. **Property-Based Testing**
   - Assume *generic properties* for reasoning models (e.g. For every observation there must be at least one diagnosis)
   - Generate tests by perturbing observations.

2. **Coverage-Directed Testing**
   - Design *novel coverage criteria* (e.g. Every diagnosis must be generated at least once)
   - Generate tests by perturbing observations.

3. **SAT-Based Testing**
   - Verify every diagnosis by using a SAT-solver.
### Miscellaneous Information

- **Tool:** [https://github.com/yavuzkoroglu/gfuzzer-release](https://github.com/yavuzkoroglu/gfuzzer-release)
- **Contact:**
  1. yavuz.koroglu@boun.edu.tr or ykoerogl@ist.tugraz.at
  2. wotawa@ist.tugraz.at

### Acknowledgement

The research was supported by ECSEL JU under the project H2020 737469 **AutoDrive**. [https://iktderzukunft.at/en/][1]

### Acknowledgement

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.
Thank You