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

Yavuz Koroglu[1] Franz Wotawa[2]

[1]Department of Computer Engineering, *Bogazici University*, TURKEY

[2]Institute for Software Technology, *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

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

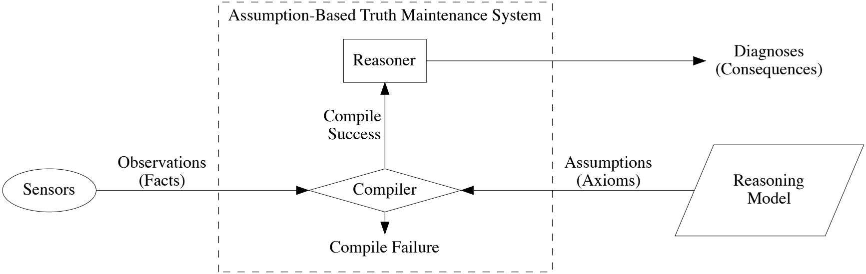


Figure: Assumption-based Truth Maintenance System (ATMS) Overview

Necessary Condition for Reasoning

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

- **Compiler testing** is required.

Our Reasoning Engine: ATMS

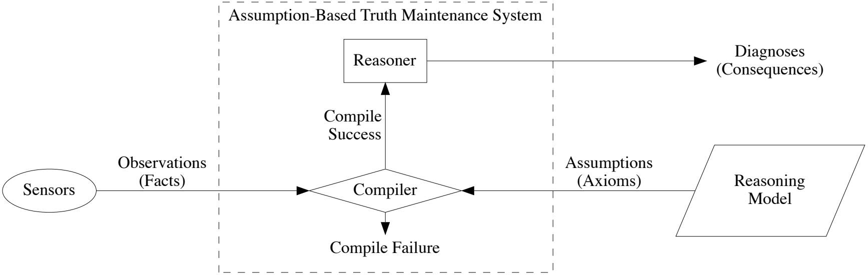


Figure: Assumption-based Truth Maintenance System (ATMS) Overview

Example (Reasoning Model)

- $\overbrace{hitGas}^{\text{observation}} \wedge \overbrace{Running}^{\text{assumption}} \rightarrow moving$
- $hitGas \wedge Broken \rightarrow notMoving$
- $moving \wedge notMoving \rightarrow \bot$

Example (Observation)

- $hitGas$
- $moving$

Our Reasoning Engine: ATMS

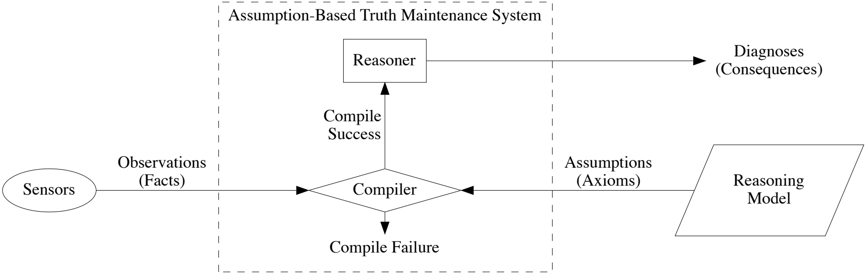


Figure: Assumption-based Truth Maintenance System (ATMS) Overview

Example (Diagnoses)

$$D = \{\{Running\}\}$$

Proceed to DEMO

Compiler Errors

Actual \ ATMS	Compile Failure	Consequence List
Reject	Test Passed	Type I Error
Accept	Type II Error	Test Passed

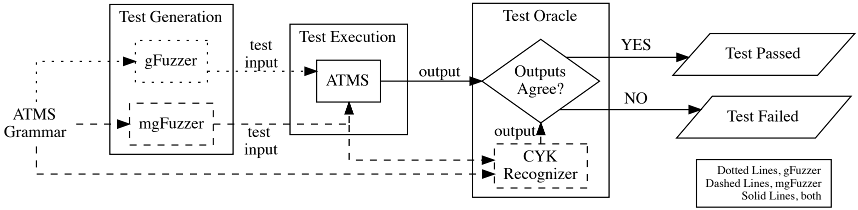
Type I Errors (False Positives)

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

Type II Errors (False Negatives)

Compiler **rejects** a **valid input**.

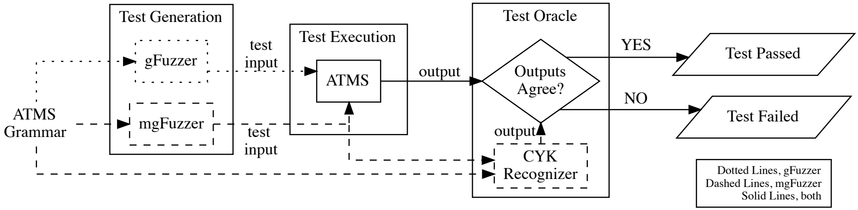
gFuzzer Overview



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

Mutation Operators I

Terminal Replacement (TR)

Swaps two non-equal terminals.

$$(\text{<A>}::= a, \text{<B>}::= b) \Rightarrow (\text{<A>}::= b, \text{<B>}::= a)$$

Example

$$(\text{<Digit>}::= 1, \text{<Op>}::=\rightarrow) \Rightarrow (\text{<Digit>}::=\rightarrow, \text{<Op>}::= 1)$$

Note

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

1. $\text{<R>}::=\text{<Q>}\text{<S>}$ and
2. $\text{<R>}::= t$

Mutation Operators II

DEletion (DE)

Replaces all expansions of a rule with empty string.

$$<\text{A}>\in G \Rightarrow <\text{A}>::= \boldsymbol{\varepsilon}$$

Example

$$(<\text{NonEmptyList}>::=<\text{Element}><\text{Rest}>) \Rightarrow <\text{NonEmptyList}>::= \boldsymbol{\varepsilon}$$

Note

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

1. $<\text{R}>::=<\text{Q}><\text{S}>$ and
2. $<\text{R}>::= t$

Mutation Operators III

DUplication (DU)

Duplicates a rule.

$<A>::=<B><C> \Rightarrow (<A>::=<A'><A'>, <A'>::=<B><C>)$

Example

$(<Add>::=<Term><PlusTerm>) \Rightarrow (<Add>::=<Add'><Add'>)$

Note

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

1. $<R>::=<Q><S>$ and
2. $<R>::= t$

Mutation Operators IV

EXchange (EX)

Swaps the order of non-terminals.

(<A>::=<B><C>) ⇒ (<A>::=<C><B>)

Example

(<Add>::=<Term><PlusTerm>) ⇒ (<Add>::=<PlusTerm><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*

Mutation Operators V

Recursion Insertion (RI)

Enables infinite recursion on a random rule.

$$(<\mathrm{A}>\in G) \Rightarrow (<\mathrm{A}>::=<\mathrm{A}> \mid <\mathrm{A}><\mathrm{A}>)$$

Example

$$(<\mathrm{False}>::= \bot) \Rightarrow (<\mathrm{False}>::= \bot \mid <\mathrm{False}><\mathrm{False}>)$$

Note

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

1. $<\mathrm{R}>::=<\mathrm{Q}><\mathrm{S}>$ and
2. $<\mathrm{R}>::= t$

Mutation Operators VI

Terminal Insertion (TI)

Inserts a random terminal to a random rule.

$$(\text{<A>} \in G) \Rightarrow (\text{<A>::=<A>}\ x) \text{ or } (\text{<A>::=}\ x\ \text{<A>})$$

Example

$$(\text{<False>::=}\ \bot) \Rightarrow (\text{<False>::=}\ \neg\bot)$$

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

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

	Failed	Passed	Total	Rule (%)	Code (%)
gFuzzer	0	1,490,388	1,490,388	100	67.6
mgFuzzer	2	1,024	1,026	100	75.9
Both	2	1,491,412	1,491,414	100	75.9

Failed Tests

Test #1	`x1,falsex2()x3->x2.` `Assumption1.`

Test #2	`Assumption1,x2(false,x3)false.`
	`Assumption3.`
	`Assumption2.`
	`x1.`

- *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
- **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/ bmvit

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