

Fuzz4All: Universal Fuzzing with Large Language Models

Chunqiu Steven Xia
University of Illinois
Urbana-Champaign, USA
chunqiu2@illinois.edu

Matteo Paltenghi
University of
Stuttgart, Germany
mattepalte@live.it

Jia Le Tian
University of Illinois
Urbana-Champaign, USA
jialelt2@illinois.edu

Michael Pradel
University of
Stuttgart, Germany
michael@binaervarianz.de

Lingming Zhang
University of Illinois
Urbana-Champaign, USA
lingming@illinois.edu

ICSE '24: Proceedings of the IEEE/ACM 46th International Conference on
Software Engineering
Article No.: 126, Pages 1 - 13

背景 ～ファジング～

ファジングとは

ファズテスト[69,84]のこと。テスト対象システム（SUT）の予期しない動作を明らかにするための入力を自動生成するテスト手法

従来ファジングツール

◆ 生成ベースのファジング[34,49,81]

対象言語の定義済み文法などを使用して、完全なコードスニペットを直接合成することを目的としている

◆ 突然変異ベースのファジング[21,31,69]

突然変異演算子または変換ルールを一連の高品質のファジングシードに適用する

ファジングシード

ファジングで使用する入力データの初期セット

課題 ～従来手法の限界～

◆ 課題 1 対象システムや言語への依存

従来のファジングツールの多くは、特定のSUTまたは特定の言語を対象に設計されている。ある SUT でうまく機能するファジング戦略が、別の SUT ではまったく機能しないことがある。

◆ 課題 2 バージョンアップ時のサポートの欠如

特定の言語やSUTのバージョン向けに設計された従来のファジングツール、新しいバージョンで有効性を失う可能性があり、新しく実装された機能をテストするために簡単には使用することができない。

◆ 課題 3 カバーできる入力空間の制限

生成ベースのファジングでは完全な言語文法のサブセットのみテストするように制限される。突然変異ベースのファジングでは突然変異演算子によって制限され、入手困難なファジングシードが必要になる。

目的・アプローチ

目的

幅広い入力言語やSUTを対象にした、汎用なファジングを行うことができるFuzz4Allを提案

アプローチ

入力生成および突然変異エンジンとしてLLMを活用

LLMは多くのプログラミング言語で大量の例を事前にトレーニングしているため、コードの構文や意図を暗黙的に理解している

2つの主要なステップに分かれる

◆ 自動プロンプト

ユーザ入力をファジシングに適したプロンプトに変換

◆ ファジングループ

多様なファジニングセットの生成

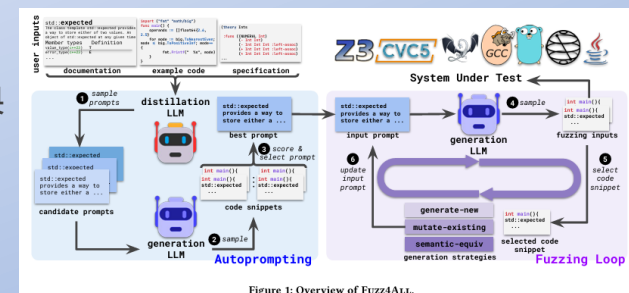
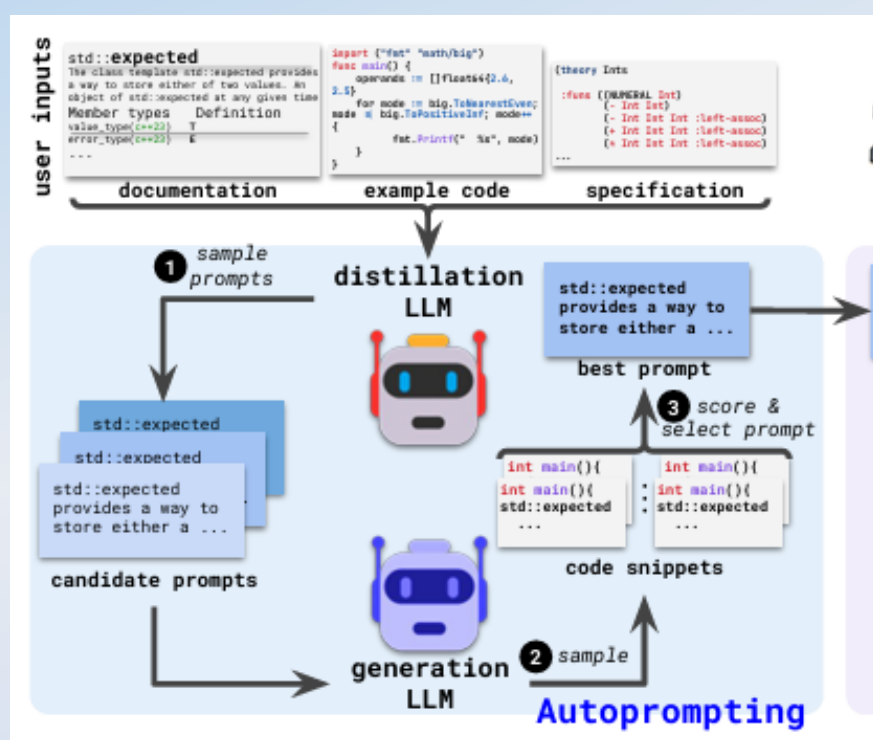


Figure 1: Overview of FUZZ4ALL.

Fuzz4All ～プロンプト自動生成～



Algorithm 1: Autoprompting for fuzzing

```

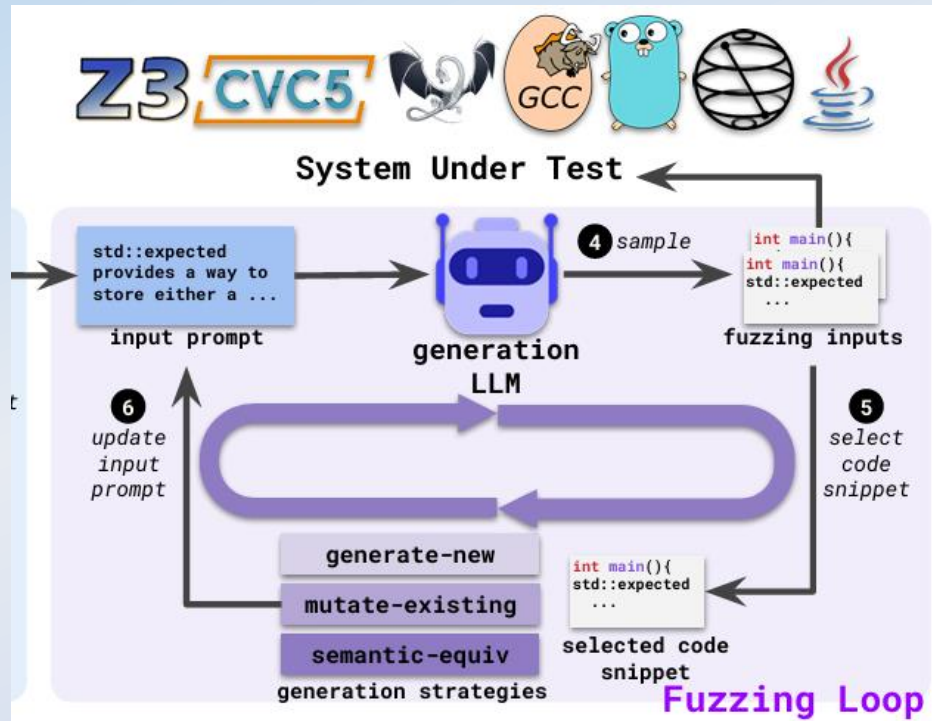
1 Function Autoprompting:
  Input : userInput, numSamples
  Output: inputPrompt
2 greedyPrompt ←  $\mathcal{M}_D$  (userInput, APIInstruction, temp=0)
3 candidatePrompts ← [ greedyPrompt ]
4 while |candidatePrompts| < numSamples do
5   prompt ←  $\mathcal{M}_D$  (userInput, APIInstruction, temp=1)
6   candidatePrompts ← candidatePrompts + [ prompt ]
7 inputPrompt ←  $\arg \max_{p \in \text{candidatePrompts}} \text{Scoring}(\mathcal{M}_G(p), \text{SUT})$ 
8 return inputPrompt
  
```

High level description of feature	The C++23 <code>std::expected</code> class template provides a way to store either an expected value of type <code>T</code> or an unexpected value of type <code>E</code> . It is useful for handling functions that may return an error or a valid result. The stored value is allocated directly within the storage occupied by the expected object, without dynamic memory allocation.
Descriptions of the inputs	The template parameters are <code>T</code> (the expected value type) and <code>E</code> (the unexpected value type). Both types must meet the Destructible requirements, and certain types are not allowed.
Different usages of target	<code>std::expected</code> provides member functions for construction, destruction, assignment, and accessing the stored values. Observers like <code>bool</code> and <code>has_value</code> can be used to check if the object contains an expected value. Functions like <code>value</code>, <code>error</code>, and <code>value_or</code> can be used to access the expected or unexpected values. Monadic operations like <code>and_then</code>, <code>transform</code>, <code>or_else</code>, and <code>transform_error</code> allow chaining operations on expected values and handling errors in a functional manner. Modifiers like <code>emplace</code> and <code>swap</code> can be used to construct the expected value in-place or exchange the contents of expected objects. Non-member functions like <code>operator==</code> and <code>swap(std::expected)</code> provide comparison and swapping functionality. Helper classes like <code>unexpected</code>, <code>bad_expected_access</code>, and <code>unexpected_t</code> are used to represent unexpected values, exceptions, and in-place construction tags for unexpected values in expected objects.

Figure 2: Autoprompting result for `std::expected`.

- ① プロンプトの抽出 ユーザ入力は冗長なので、候補プロンプトとして品質の高いプロンプトを抽出する (Figure 2)
- ② スニペットの作成 候補プロンプトからコードスニペットを作成する
- ③ スコアリング 最近のNLP研究[87]のスコアリング関数を使用し、有効なコードスニペットを選択

Fuzz4All ～ファジンググループ～



Algorithm 2: Fuzzing loop

```
1 Function FuzzingLoop:
  Input : inputPrompt, timeBudget
  Output: bugs
2  genStrats  $\leftarrow$  [ generate-new, mutate-existing,
  semantic-equiv ]
3  fuzzingInputs  $\leftarrow$   $M_G$  (inputPrompt + generate-new)
4  bugs  $\leftarrow$  Oracle (fuzzingInputs, SUT)
5  while timeElapsed < timeBudget do
6    example  $\leftarrow$  sample (fuzzingInputs, SUT)
7    instruction  $\leftarrow$  sample (genStrats)
8    fuzzingInputs  $\leftarrow$   $M_G$  (inputPrompt + example +
9    instruction)
10   bugs  $\leftarrow$  bugs + Oracle (fuzzingInputs, SUT)
11 return bugs
```

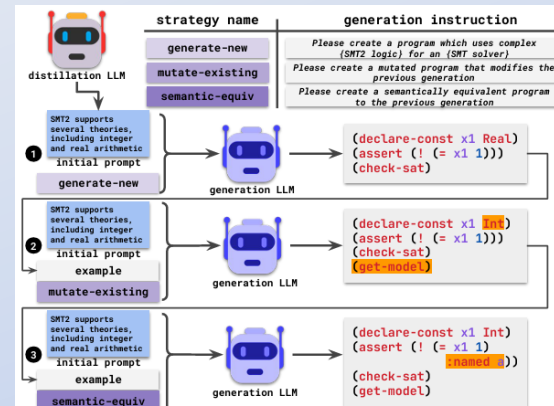


Figure 3: Fuzzing strategies and example of fuzzing loop.

- ④スニペットの作成 プロンプトからコードスニペットを作成する
- ⑤ファジングを選択 ランダムなファジング入力を選択する
- ⑥プロンプトを生成 3つの生成戦略により、新たなプロンプトを生成する

Fuzz4AIの汎用性評価

6パターンを想定し，ベースラインと比較

Table 1: SUTs and baseline tools.

Language	SUT(s)	Baseline tool(s)	Version
C	GCC, Clang	GRAYC [21], CSMITH [81]	GCC-13.1.1
C++	G++, Clang++	YARPGEN [49]	G++-13.1.1
SMT2	Z3, CVC5	TYPEFUZZ [59]	CVC5-1.0.5
Go	Go	GO-FUZZ [25]	go-1.20.6
Java	javac	HEPHAESTUS [11]	OpenJDK-javac-18
Python	Qiskit	MORPHQ [58]	qiskit-0.43.1

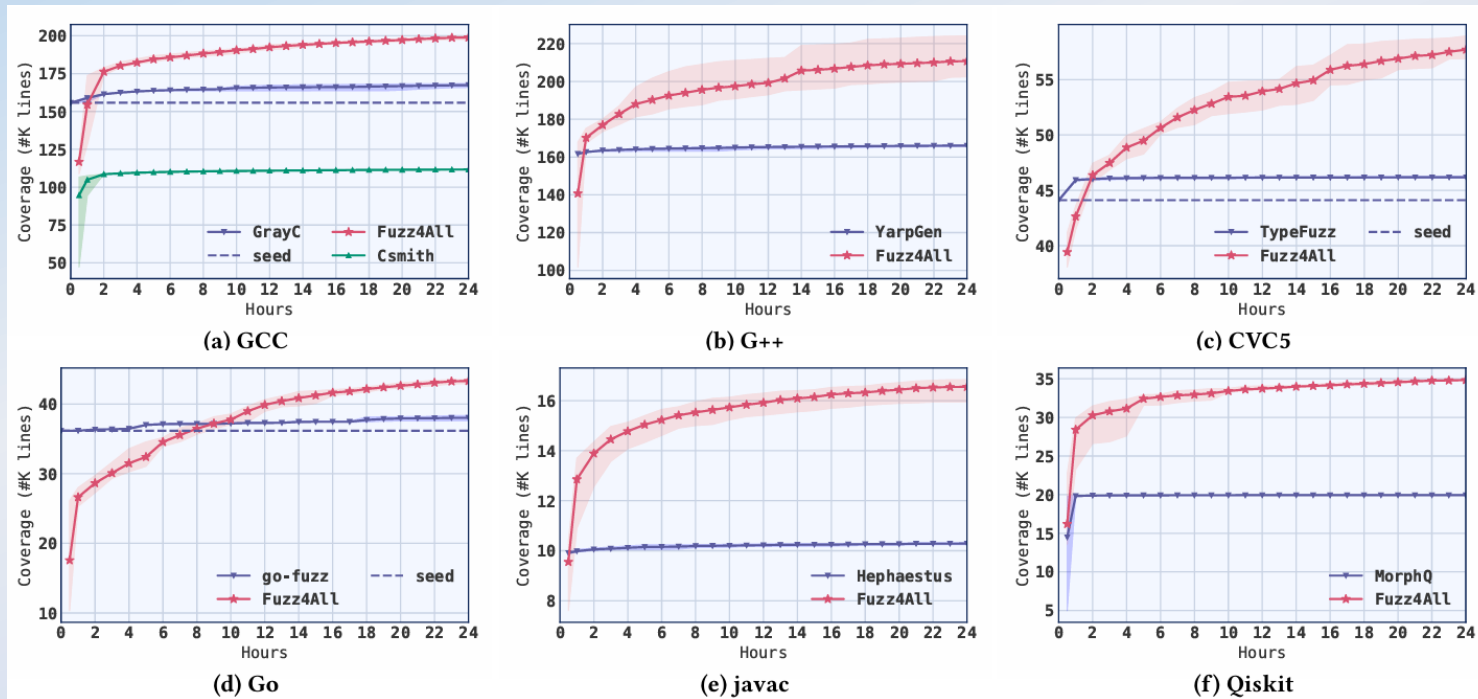
C, C++, Javaはコンパイラ

SMT2はSMTソルバー（命題論理の充足可能性検査）

Goはツールチェーン

Pythonは量子コンピューティングプラットフォーム

RQ1 ベースラインと比較



24時間で平均36.8%の改善が見られた

YarpGenやMorphQなどの生成ベースのベースラインと比較して、Fuzz4Allは即座に高いカバレッジを出した

go-fuzzやGrayCなどの突然変異ベースのベースラインと比較して、Fuzz4Allはゆっくり確実に多くのコードをカバーできた

RQ1 ベースラインと比較

Table 2: FUZZ4ALL against state-of-the-art fuzzers (* indicates statistically significant coverage improvement).

Target	Fuzzer	# programs	% valid	Coverage
GCC	GRAYC	104,326	95.96%	167,453
	CSMITH	61,883	99.99%	111,668
	FUZZ4ALL	44,324	37.26%	*198,927 +18.8%
G++	YARPGEN	255,581	99.99%	166,614
	FUZZ4ALL	26,365	40.74%	*210,743 +26.5%
CVC5	TYPEFUZZ	43,001	93.24%	46,174
	FUZZ4ALL	36,054	47.63%	*57,674 +24.9%
Go	GO-FUZZ	20,002	100.00%	38,024
	FUZZ4ALL	22,817	23.02%	*43,317 +13.7%
javac	HEPHAESTUS	728,217	57.22%	10,285
	FUZZ4ALL	31,967	49.05%	*16,552 +60.9%
Qiskit	MORPHQ	38,474	100.00%	19,929
	FUZZ4ALL	33,454	24.90%	*34,988 +75.6%

#programs : ユニークな入力の数

#valid : 有効な入力の割合

#Coverage : 最終カバレッジと
ベースラインに対する改善

入力数と有効率が低いにもかかわらずFuzz4Allは高いカバレッジを出した

その他の評価

◆ RQ2 ターゲットファジングの有効性

特定の機能に絞ったファジング入力の生成について, Fuzz4Allのヒット率の平均は83.0%であった

◆ RQ3 部分的な評価

自動プロンプトとファジンググループのそれぞれについて, もう片方のステップの影響を少なくすることによって評価

◆ RQ4 バグの発見

今回調査した9つのSUTで, Fuzz4Allは98個のバグを発見し, そのうち64個は開発者に知られていなかった

発見されたバグについて

Table 5: Summary of FUZZ4ALL-detected bugs.

	Total	Confirmed		Pending	Won't fix
		Unknown	Known		
GCC	30	14	11	5	0
Clang	27	18	9	0	0
CVC5	9	7	2	0	0
Z3	14	12	0	0	2
Go	4	2	2	0	0
Java	3	3	0	0	0
Qiskit	11	8	2	1	0
Total	98	64	26	6	2

```
#include <optional>
void y(std::optional<int> z)
    noexcept(noexcept(std::optional<int>{z})) {}
```



(a) GCC bug: Internal compiler error (segmentation fault)

```
#include <iostream>
using E = std::numeric_limits<int>;
auto fail(E e) → decltype(throw e, void()) { throw e; }
```



(b) Clang bug: Segmentation fault

```
package main
import ("runtime")
func main() { runtime.ReadMemStats(nil) }
```



(c) Go bug: Segmentation violation

```
from qiskit import QuantumCircuit, ClassicalRegister
crz = ClassicalRegister(1, name="crz")
qc = QuantumCircuit(crz)
qc.qasm(filename="my.qasm")
QuantumCircuit.from_qasm_file("my.qasm")
```



(d) Qiskit bug: Crash

Figure 5: Exemplary bugs found by FUZZ4ALL.

結論・所感

結論

Fuzz4Allは最先端のツールと比較してカバレッジを大幅に向上できることが実証された

所感

今回はベースラインと提案手法を別々に使用していたが、ベースラインと併用した際にカバレッジが高くなるのかが気になった

LLMをかなり有効活用した例であると思った

拡大 ～Fuzz4Allの概要～

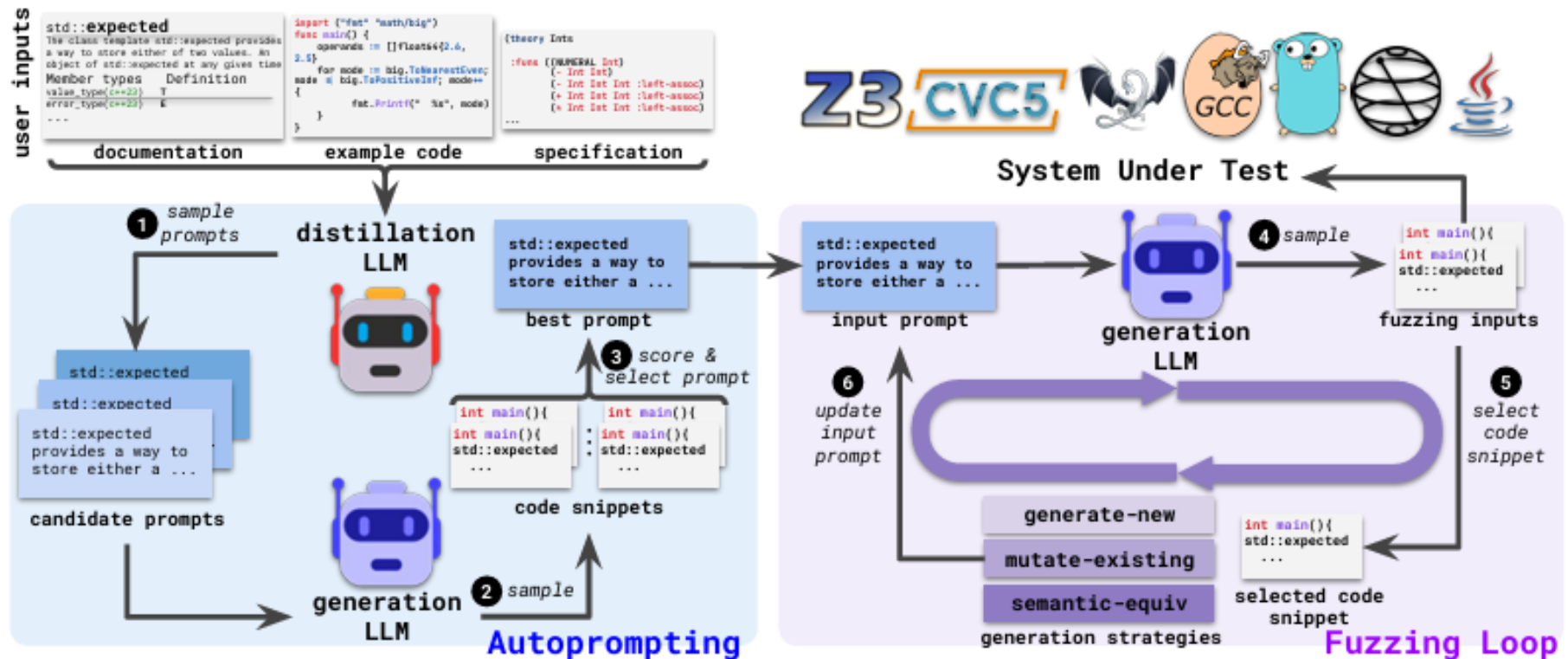


Figure 1: Overview of Fuzz4ALL.

拡大 ～プロンプト自動生成～

Algorithm 1: Autoprompting for fuzzing

```
1 Function Autoprompting:
   Input : userInput, numSamples
   Output: inputPrompt
2   greedyPrompt  $\leftarrow M_D$  (userInput, APInstruction, temp=0)
3   candidatePrompts  $\leftarrow$  [ greedyPrompt ]
4   while |candidatePrompts| < numSamples do
5     prompt  $\leftarrow M_D$  (userInput, APInstruction, temp=1)
6     candidatePrompts  $\leftarrow$  candidatePrompts + [ prompt ]
7   inputPrompt  $\leftarrow \arg \max_{p \in \text{candidatePrompts}} \text{Scoring}(M_G(p), \text{SUT})$ 
8   return inputPrompt
```

High level description of feature

The C++23 `std::expected` class template provides a way to store either an expected value of type `T` or an unexpected value of type `E`. It is useful for handling functions that may return an error or a valid result. The stored value is allocated directly within the storage occupied by the expected object, without dynamic memory allocation.

Descriptions of the inputs

The template parameters are `T` (the expected value type) and `E` (the unexpected value type). Both types must meet the Destructible requirements, and certain types are not allowed.

Different usages of target

`std::expected` provides member functions for construction, destruction, assignment, and accessing the stored values. Observers like operator `bool` and `has_value` can be used to check if the object contains an expected value. Functions like `value`, `error`, and `value_or` can be used to access the expected or unexpected values.

Monadic operations like `and_then`, `transform`, `or_else`, and `transform_error` allow chaining operations on expected values and handling errors in a functional manner.

Modifiers like `emplace` and `swap` can be used to construct the expected value in-place or exchange the contents of expected objects. Non-member functions like `operator=` and `swap(std::expected)` provide comparison and swapping functionality.

Helper classes like `unexpected`, `bad_expected_access`, and `unexpected_t` are used to represent unexpected values, exceptions, and in-place construction tags for unexpected values in expected objects.

Figure 2: Autoprompting result for `std::expected`.

拡大 ～ファジingroupグループ～

Algorithm 2: Fuzzing loop

```

1 Function FuzzingLoop:
  Input :inputPrompt, timeBudget
  Output: bugs
2  genStrats  $\leftarrow$  [ generate-new, mutate-existing,
3    semantic-equiv ]
4  fuzzingInputs  $\leftarrow$   $\mathcal{M}_{\mathcal{G}}$  (inputPrompt + generate-new)
5  bugs  $\leftarrow$  Oracle (fuzzingInputs, SUT)
6  while timeElapsed < timeBudget do
7    example  $\leftarrow$  sample (fuzzingInputs, SUT)
8    instruction  $\leftarrow$  sample (genStrats)
9    fuzzingInputs  $\leftarrow$   $\mathcal{M}_{\mathcal{G}}$  (inputPrompt + example +
10   instruction)
11   bugs  $\leftarrow$  bugs + Oracle (fuzzingInputs, SUT)
12 return bugs
  
```

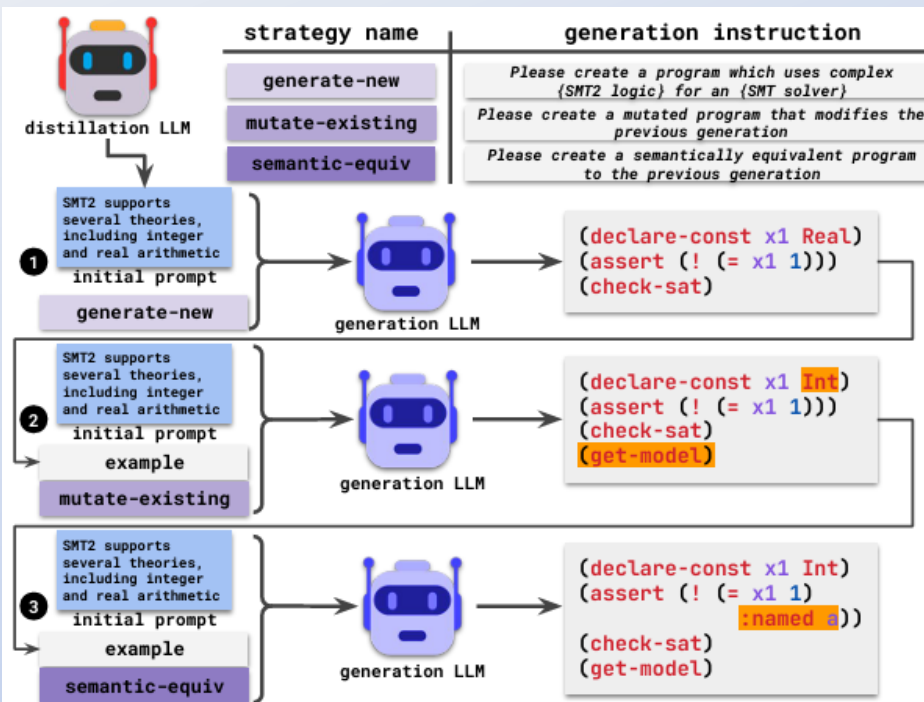


Figure 3: Fuzzing strategies and example of fuzzing loop.

略語集

- LLM : Large Language Model / 大規模言語モデル
- SUT : System Under Test / テスト対象システム
- NLP : Natural Language Processing / 自然言語処理