Reliable Code Generation with LLMs via Resonance Check

Sergey Mechtaev

Peking University

2025

Preliminaries: Code Generation

- R are natural language (NL) requirements;
- m is an LLM, $m(\cdot \mid R)$ is a distribution of programs;
- Programs $p \sim m(\cdot \mid R)$ sampled based on requirements are functions

$$A_1 \times \cdots \times A_n \to B$$

where $A_1, \cdots, A_n$ are input domains, B is the output domain.

Prompt

Implement a function `f(n: int) -> int` that computes the n-th Fibonacci number.

Response

```
def f(n: int) -> int:
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a
```

Preliminaries: Specification

The semantics of NL requirement $[\![\cdot]\!] : \mathrm{NL} \to 2^{A_1 \times \cdots \times A_n \times B}$ is a specification implicitly defined by these requirements, e.g.

$$[\![\text{“}\textit{multiply two numbers}\text{”}]\!] = \{(2,2,4),(2,3,6),\cdots\}.$$

For simplicity, assume that all inputs and outputs are valid and useful:

$$\forall a_1 \in A_1,\ \cdots, a_n \in A_n\ \exists b \in B.\ (a_1,\cdots,a_n,b) \in [\![R]\!]$$
$$\forall b \in B\ \exists a_1 \in A_1,\ \cdots, a_n \in A_n.\ (a_1,\cdots,a_n,b) \in [\![R]\!]$$

A program $p \sim m(\cdot \mid R)$ is *correct* iff

$$\forall\ a_1 \in A_1,\ \cdots, a_n \in A_n.\ (a_1,\cdots,a_n,p(a_1,\cdots,a_n)) \in [\![R]\!],$$

denoted as $[\![R]\!] \vdash p$.

It is *incorrect* iff

$$\exists a_1 \in A_1,\ \cdots, a_n \in A_n.\ (a_1,\cdots,a_n,p(a_1,\cdots,a_n)) \notin [\![R]\!],$$

denoted as $[\![R]\!] \nvdash p$.

Exactness of Requirements/Specifications

Definition (Exact Specification)

A specification is *exact* if it is a function. Otherwise, it is *inexact*.

Example (Inexact Requirements)

These requirements permit multiple outputs for the same input:

$$\begin{aligned}&[\![\text{“}\textit{return a positive element of a given set}\text{”}]\!] = \\ &\quad \{\ (\{1,-1,3\},1),\\ &\quad\ \ (\{1,-1,3\},3),\cdots\}.\end{aligned}$$

Describing Specifications

Assume any specification can be expressed in natural language using

$$(\!|\cdot|\!) : 2^{A_1 \times \cdots \times A_n \times B} \to \mathrm{NL},$$

e.g.

$$(\!|\, \{\, (0,1), (1,2), (2,3), \cdots \,\}\, |\!) =$$
"*increment a given number*"

Reliability of LLM-Generated Code

LLMs make mistakes, e.g. hallucinate [1], which results in incorrect programs.

Challenge: how to automatically catch mistakes without explicit $[\![R]\!]$?

Prompt

You are given a positive integer array ‘nums‘. Return the total frequencies of elements in ‘nums‘ such that those elements all have the maximum frequency.

Response

```
def count(nums):
  freq = Counter(nums)
  max = freq.values()
  count = len([
   k for k, v in
   freq.items()
    if v == max
  ])
  return count # missing `* max`
```

Previous Work: Consistency 一致性

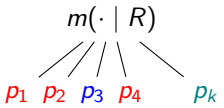
Definition

Programs p_1 and p_2 are equivalent ($p_1 \equiv p_2$) iff

$$\forall a_1 \in A_1, \cdots, a_n \in A_n.\ p_1(a_1, \cdots, a_n) = p_2(a_1, \cdots, a_n).$$

CodeT [2] partitions samples into equivalence classes:

采样 - 根据输出划分等价类



$p_1 \equiv p_2 \equiv p_4 \equiv \cdots$

$p_3 \equiv \cdots$

$p_k \equiv \cdots$

Then, it selects a sample, say, p_1, if $[p_1]$ is the largest equivalence class.

最大等价类

Limitations:

- LLMs consistently make the same mistakes.
- The method does not work for inexact requirements.

Previous Work: Confidence 置信度

Measure confidence of an LLM response through token probabilities or verbalized self-ask [3].

Problem: Confidence measures weakly correlate with correctness.

Prompt

Implement a Fibonacci function in Python and print how confident you are about its correctness (from 0% to 100%).

Response

```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    a, b = 0, 1
    for _ in range(2, n + 1):
        a, b = b, a + b
    return b
```

Confidence: 99%. The 1% uncertainty accounts for potential uncaught edge cases.

Previous Work: Assertions

Sample both programs and assertions specifying the expected output [4], i.e. executable predicates directly capturing $[\![R]\!]$.

Intuition: for some problems, it is easier to check correctness of an output than to compute it.

易检查
难计算

Prompt

The function s(l: list) -> list sorts the list in non-descending order. Create a predicate that checks, for a given l, if s(l) returns a correct output.

Response

```
def is_sorted_correctly(l: list) -> bool:
    sorted_l = s(l)
    if Counter(l) != Counter(sorted_l):
        return False
    for i in range(len(sorted_l) - 1):
        if sorted_l[i] > sorted_l[i + 1]:
            return False
    return True
```

Problems with Assertions

- For some problems, an implementation is the simplest description of the expected output.
- Given a function $f : X \to Y$ implementing R and a predicate $p : X \times Y \to \{0, 1\}$ implementing $\widetilde{R}$ (requirements for an assertion that checks R), we only have a weak guarantee that

$$\forall x \in X.\ p(x, f(x))$$

implies

$$[\![\widetilde{R}]\!] \vdash p \Rightarrow [\![R]\!] \vdash f.$$

Prompt

Function $F(n)$ computes the n-th Fibonacci number. Implement a predicate in Python that checks for a given n if the output of F is correct.

Response

```python
def correct_fib(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a

def is_fib_correct(n, F):
    try:
        return F(n) == correct_fib(n)
    except:
        return False
```

New Idea: Resonance

- Sample several *resonator* programs with deliberately altered requirements.
- Check their *resonance property* that provably holds iff either all programs correctly implement their requirements, or all programs wrongly implement their requirements.
- If the programs *resonate*, then they are either correct, or have *sympathetic faults*, i.e. low-probability faults that cancel each other out.

Definition

Let $r_1 \sim m(\cdot \mid R_1), \cdots, r_n \sim m(\cdot \mid R_n)$ be resonators. A hyperproperty ϕ over $r_1, \cdots, r_n$ is a resonance property if it holds iff either all programs are correct or all programs are incorrect:

$$\phi[r_1, \cdots, r_n] \Leftrightarrow \bigwedge_i [\![R_i]\!] \vdash r_i \vee \bigwedge_i [\![R_i]\!] \nvdash r_i.$$

Consistency is A Trivial Resonance Property

不重要的. 个数不足道的. 琐碎的.

Proposition (Resonance Generalizes Consistency)

Let R ($R_1, R_2, \cdots$) be requirements, and $r_1, r_2 \sim m(\cdot \mid R)$ be resonators generated based on these requirements. Then, the following is a resonance property:

$$r_1 \equiv r_2.$$

Problems:

- sympathetic faults are high-probability, because LLMs consistently make the same mistakes.
- likely to fail for correct programs when the requirements are inexact.

Better Resonance Properties

To account for different types of specifications, check several resonance properties in the order of raising complexity and generality:

- FOR-INV: Forward function + (Partial) Inverse
- FOR-FIB: Forward function + (Partial) Fiber function
- COR-FIB: Correspondence function + (Partial) Fiber function
- VAL-FIB: Validator function + Its (Partial) Fiber function

FOR-INV Resonance Property

给n. 请计算第n个斐波那契

Definition (Inverse Specification)

Let $S \subset A \times B$ be a specification. Its inverse is $S^{-1} \triangleq \{(b, a) \mid (b, a) \in S\}$. Similarly, the inverse of requirements R is $R^{-1} \triangleq (\!| \, [\![R]\!]^{-1} \, |\!)$. For multi-argument functions, the inverse should produce a tuple as the output.

Proposition (FOR-INV)

Let R be requirements, $f : A \to B \sim m(\cdot \mid R)$ and $g : B \to A \sim m(\cdot \mid R^{-1})$ be resonators. The following is a resonance property:

$$g \circ f \equiv \mathrm{id_A} \wedge f \circ g \equiv \mathrm{id_B}.$$

Hypothesis: inversion-based resonance reduces sympathetic bugs compared to the standard consistency, because f and g require different algorithms.
Problem: likely to fail if $[\![R]\!]$ is not a bijection.

Liberal Resonance Property

Problem: it might be too expensive to check the right inverse property $f \circ g \equiv \mathrm{id}_B$ on the entire set B.

Idea: since not all resonators are equally important, we can weaken the property to make it easier to check.

Definition (Liberal FOR-INV)

Let R be requirements, $f : A \to B \sim m(\cdot \mid R)$ and $g : B \to A \sim m(\cdot \mid R^{-1})$ be resonators. The left inverse property

$$g \circ f \equiv \mathrm{id}_{\mathrm{A}}$$

is not a resonance property since, although it checks f, it captures the behavior of g only on the range of f. We call it a liberal resonance property.

Limitations of FOR-INV

Prompt

Implement `f(i: int) -> int` that returns a number greater than i by at most 2.

Response

```
def f(i: int) -> int:
  return i+1
```

Prompt

Implement `g(i: int) -> int` that returns a number smaller than i by at most 2.

Response

```
def g(i: int) -> int:
  return i-2
```

Independently generated forward and backward functions for inexact requirements may not match. **Note:** the generation must be independent to reduce the chance of sympathetic faults.

FOR-FIB Resonance Property

Definition (Fiber Specification)

Let $S \subset A \times B$ be a specification. Its fiber function specification is

$$S^{\succ} \triangleq \left\{ (b, \{a \mid (a,b) \in S\}) \;\middle|\; (\cdot, b) \in S \right\}.$$

Given requirements R, the requirements for a fiber function are $R^{\succ} \triangleq (\!| \, [\![R]\!]^{\succ} \, |\!)$.

Proposition (Liberal FOR-FIB)

Let R be requirements, $f : A \to B \sim m(\cdot \mid R)$ and $g : B \to 2^A \sim m(\cdot \mid R^{\succ})$ be resonators. The following is a liberal resonance property:

$$\forall b \in B.\ f(g(b)) = \{b\} \wedge \forall a \in A.\ a \in g(f(a)),$$

since it checks for the exhaustiveness of fibers only on the range of f.

Limitations of Inverse and Fiber Resonators

Inverse functions may enumerate inputs of the forward function.

Prompt

Implement a Python function that for a given Fibinacci number n returns its index in the sequence.

Response

```
def find_fibonacci_index(n):
    if n < 0:
        return -1
    if n == 0:
        return 0
    if n == 1:
        return 1

    a, b = 1, 1
    index = 2
    while b < n:
        a, b = b, a + b
        index += 1
    return index if b == n else -1
```

Hard to compute fibers for multi-arguments functions.

Prompt

Implement a Python function that for a boolean 'found' enumerates all pairs of a list and an integer such that the list contains the integer iff the boolean is true.

Response

```
def enumerate_pairs(found):
  max_list=3
  max_int=3
  pairs = []
  ints = range(max_int + 1)
  for l in range(max_list + 1):
    for lst in product(ints, repeat=l):
      lst = list(lst)
        for num in ints:
          if (num in lst) == found:
            pairs.append(
              (lst.copy(), num))
  return pairs
```

Partial Inverses and Fiber Functions

Definition

Given a specification $S \subset A_1 \times A_2 \times \cdots \times A_n \times B$, its *partial inversion* w.r.t. the i-th argument is a specification

$$S_i^{-1} \subset A_1 \times \cdots A_{i-1} \times \mathbf{B} \times A_{i-1} \times \cdots \times A_n \times \mathbf{A_i}$$

s.t. $(a_1, \cdots, a_n, b) \in S$ iff $(a_1, \cdots, a_{i-1}, b, a_{i+1}, \cdots, a_n, a_i) \in S_i^{-1}$. Partial fiber functions are defined respectfully.

Example

Assume a specification is a function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ defined as $f(x, y) = x \mod y$. A possible partial inverse w.r.t. the first argument is $g(z, y) = y + z$; the partial fiber function is $g(z, y) = \{i * y + z\}_{i \in \mathbb{N}}$.

FOR-INV with Partial Inverses

Proposition

Let R be requirements specifying functions over two arguments, $f \sim m(\cdot \mid R)$ and $g \sim m(\cdot \mid R_1^{-1})$ be resonators. The following is a liberal resonance property (checks g on a subdomain):

$$\forall(x, y, \cdot) \ \in [\![R]\!].\ g(f(x, y), y) = x.$$

FOR-FIB with Partial Fiber Functions

Proposition

Let R be requirements, defining a specification over two arguments, $f \sim m(\cdot \mid R)$ and $g \sim m(\cdot \mid R_1^{\succ})$ be resonators. Then, the following is a liberal resonance property (checks g on a subdomain):

$$\forall(x, y, \cdot) \ \in [\![R]\!].\ f(g(f(x, y), y), y) = \{f(x, y)\} \wedge x \in g(f(x, y), y).$$

Challenge: Inexact Requirements

If the requirements are inexact, then the check $f(g(f(x,y),y),y) = \{f(x,y)\}$ may fail. For example, for the problem description

> *Implement a function that returns the list of all positive values stored in the nodes of a given graph.*

f may assume an implementation that traverses the graph in the depth-first search order, and g may assume an implementation that traverses the graph in the breadth-first search order. Although both are valid solutions, the order of elements in the output list will be different.

COR-FIB Resonance Property

Definition (Correspondence Functions)

Given a specification $S \subset A_1 \times A_2 \times \cdots \times A_n \times B$, its *correspondence* function specification is $S^{\prec} \subset A_1 \times \cdots \times A_n \times 2^B$ s.t.

$$S^{\prec} \triangleq \Big\{ (a_1, \cdots, a_n,\ \{b \mid (a_1, \cdots, a_n, b) \in S\}) \ \Big|\ (a_1, \cdots, a_n, \cdot) \in S \Big\}.$$

Given R, the requirements for a correspondence function are $R^{\prec} \triangleq (\!|\ [\![R]\!]^{\prec}\ |\!)$.

Proposition (COR-FIB)

Let R be requirements, $f : A \to 2^B \sim m(\cdot \mid R^{\prec})$ and $g : B \to 2^A \sim m(\cdot \mid R^{\succ})$ be resonators. The following is a resonance property:

$$\forall b \in B.\ (g(b) = g(f(g(b))) \wedge b \in f(g(b)) \wedge \forall a', a'' \in g(b).\ f(a') = f(a'')) \wedge$$
$$\forall a \in A.\ (f(a) = f(g(f(a))) \wedge a \in g(f(a)) \wedge \forall b', b'' \in f(a).\ g(b') = g(b'')).$$

Note: this property assumes that there is a one-to-one correspondence between equivalence classes of inputs and outputs.

Challenge: Infinite Images and Fibers

The introduced fiber and correspondence functions have to be exhaustive. This is often impossible, e.g. inverting `contains(list, element)`:

Prompt

Write a Python function that, given a list if integers, and a boolean value "include", returns all elements that are in the list if "include" is true, and all possible elements that are not in the list if "include" is false.

Response

```python
def filter_elements(lst, include):
    if include:
        return lst.copy()
    else:
        if not lst:
            return []
        minval = min(lst)
        maxval = max(lst)
        # DeepSeek-v3 assumes a subset of input domain:
        possible_elements = set(range(minval, maxval + 1))
        elements_not_in_list = list(possible_elements - set(lst))
        return elements_not_in_list
```

VAL-FIB Resonance Property

Definition (Validator Functions)

Given a specification $S \subset A_1 \times A_2 \times \cdots \times A_n \times B$, its *validator* function specification is $\widetilde{S} \subset A_1 \times \cdots \times A_n \times B \times \{0,1\}$ s.t.

$$\begin{aligned}\widetilde{S} \triangleq &\{(a_1, \cdots, a_n, b, 1) \mid (a_1, \cdots, a_n, b) \in S\} \wedge \\ &\{(a_1, \cdots, a_n, b, 0) \mid (a_1, \cdots, a_n, b) \notin S\}.\end{aligned}$$

Given R, the requirements for a validator function are $\widetilde{R} \triangleq (\!| \widetilde{[\![R]\!]} |\!)$.

Definition (VAL-FIB)

VAL-FIB for R is FOR-FIB for $\widetilde{R}$.

Note: this is for infinite images, but small input domains.

References I

[1] Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, Li Zhang, Zhongqi Li, and Yuchi Ma.
Exploring and evaluating hallucinations in llm-powered code generation.
arXiv preprint arXiv:2404.00971, 2024.

[2] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen.
Codet: Code generation with generated tests.
arXiv preprint arXiv:2207.10397, 2022.

[3] Claudio Spiess, David Gros, Kunal Suresh Pai, Michael Pradel, Md Rafiqul Islam Rabin, Amin Alipour, Susmit Jha, Prem Devanbu, and Toufique Ahmed.
Calibration and correctness of language models for code.
arXiv preprint arXiv:2402.02047, 2024.

References II

[4] Darren Key, Wen-Ding Li, and Kevin Ellis.
Toward trustworthy neural program synthesis.
arXiv preprint arXiv:2210.00848, 2022.